SANDIA REPORT
SAND2000-1444
Unlimited Release
Printed June 2000

Code Verification by the Method of Manufactured Solutions

Kambiz Salari and Patrick Knupp

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/ordering.htm

SAND2000 - 1444
Unlimited Release
Printed June 2000

Code Verification by the Method of Manufactured Solutions

Kambiz Salari
Aerosciences and Compressible Fluid Mechanics Department

Patrick Knupp
Parallel Computing Sciences Department

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0825

Abstract

A procedure for code Verification by the Method of Manufactured Solutions (MMS) is presented. Although the procedure requires a certain amount of creativity and skill, we show that MMS can be applied to a variety of engineering codes which numerically solve partial differential equations. This is illustrated by detailed examples from computational fluid dynamics. The strength of the MMS procedure is that it can identify any coding mistake that affects the order-of-accuracy of the numerical method. A set of examples which use a blind-test protocol demonstrates the kinds of coding mistakes that can (and cannot) be exposed via the MMS code Verification procedure. The principle advantage of the MMS procedure over traditional methods of code Verification is that code capabilities are tested in full generality. The procedure thus results in a high degree of confidence that all coding mistakes which prevent the equations from being solved correctly have been identified.

Intentionally Left Blank

Table of Contents

List of Figures

List of Tables

Intentionally Left Blank

1 Introduction

The subject of this report is the Verification of computer codes that simulate physical phenomena such as fluid dynamics, heat flow, groundwater flow, porous media transport, magneto-hydrodynamics, and structural mechanics by solving a system of partial differential equations using some ordered discrete approximation method (e.g., finite elements, boundary elements, finite volume, and finite difference methods). Code Verification is important because it indicates whether or not, as F. Blottner and others have so aptly put it, "the codes are solving their respective equations right." Experienced analysts and developers know better than to uncritically accept whatever numbers are given by a computer code. They are painfully aware of the many things that can go wrong. Less clear, however, is the answer to the question: how can one systematically show that codes are correct? A consensus among computational engineers and scientists on this question does not appear to have been reached. We argue in this report that a formal *procedure* known as the Method of Manufactured Solutions (MMS) can be applied to rigorously verify computer codes.

The definition of code Verification differs among authorities (Roache [1], Oberkampf [2], and Oberkampf and Blottner [3]). The view presented in this report adopts the Roache definition of code Verification [1]:

> *"The [code] author defines precisely what continuum partial differential equations and continuum boundary conditions are being solved, and convincingly demonstrates that they are solved correctly, i.e., usually with some order of accuracy, and always consistently, so that as some measure of discretization (e.g. the mesh increments) $\Delta \rightarrow 0$, the code produces a solution to the continuum equations; this is Verification."*

This definition constrains the Blottner definition of code Verification to mean that if the observed discretization error decreases to zero as the mesh increments decrease to zero, then the equations are 'solved correctly,' In other words, code Verification is a procedure to demonstrate that the governing equations, as implemented in the code, are solved consistently. As will be seen, we recommend that, when possible, one should demonstrate that the equations are solved to the theoretical order-of-accuracy of the discretization method.

By this definition, code Verification is not concerned with asking the question, are we solving the right equation? The is part of code Validation. Nor is code Verification concerned with asking if the best (i.e., most efficient and robust) numerical method has been used.

It is important to distinguish between the Verification of *codes* and the Verification of *calculations* [1]. In the definition of Roache, code Verification does not concern itself with whether or not a specific calculation using the code is accurate. Code Verification, which in Roache's definition, involves Verification of the order-of-accuracy of the discretized equations, need be performed only once[1] and from that point on, one can claim the code is solving its equations correctly. In contrast, Verification of calculations (also referred to as solution Verification) is concerned with estimating the overall *magnitude (not order)* of the discretization error of a particular calculation. Solution Verification methodology is an on-going process which should be applied to every set of calculations. Unfortunately, both of these activities use the word Verification, which creates

1. Unless the code is modified, in which case Verification must be repeated.

confusion. *To solve this problem, we propose the term Solution Accuracy Assessment (SAA) in place of solution Verification or Verification of calculations.*

In the AIAA definition of Verification [2], code Verification and solution Verification are lumped together. We advocate that these two types of Verification should both be performed, but independently of one another (see also [1]). It frequently happens in practice that SAA is performed on calculations using unverified codes. In this case, if the code contains a coding mistake that affects the solution, then the observed order-of-accuracy of the calculation (not the theoretical order) may be incorrect. This can result in under-estimation of the discretization error. It is clear that code Verification is a pre-requisite to SAA. Furthermore, because SAA is tied to a particular calculation, it should not be preformed simultaneously with code Verification.

The idea of manufacturing solutions to PDE's has been known for a long time. Only in the last two decades has the use of manufactured solutions been suggested as a means of code Verification. One of the earliest published references to using manufactured solutions[1] to identify coding mistakes can be found in Shih [5]. Although the paper was highly original, a major deficiency in that reference was that there was no mention of the use of grid refinement to identity the order-of-accuracy. The idea of coupling the use of manufactured solutions with mesh refinement for estimating the order-of-accuracy is due to Steinberg and Roache [6], Roache et.al. [7], and Roache [8]. This report generally follows the pioneering work of Roache (see reference [1] for a general discussion). The present authors have Verified nearly a dozen codes by the MMS procedure and have developed many details of the method. Our main purpose in this report is to describe the MMS procedure by explaining in detail the art of applying MMS to Verification of engineering software.

In this Section, we defined code Verification and distinguish it from other related software and analysis activities. In Section 2 we discuss fundamentals of code Verification, describe traditional methods of code testing, and define various types of coding mistakes. Section 3 defines and describes manufactured solutions and provides guidelines for their construction. Section 4 reviews various measures of discretization error and describes a method by which the order-of-accuracy of a numerical algorithm may be determined from computed results. Section 5 describes the MMS code Verification procedure in detail. Section 6 provides examples from computational fluid dynamics that illustrate how the MMS procedure is applied. Section 7 present a series of blind-tests which illustrate that the MMS procedure is capable of detecting any coding mistake which prevents the equations from being solved correctly. Section 8 summarizes the strengths and limitations of the MMS code Verification procedure. Conclusions and recommendations are given in Section 9.

1. The term *manufactured solution* was first used by Oberkampf, et.al. [4] in the context of code Verification but the procedure was not further developed.

2 Fundamentals of Code Verification

2.1 Governing Equations

Development of computer codes that simulate physical phenomena by means of ordinary differential equations (ODEs) or partial differential equations (PDEs) is based on a sequence of prerequisite activities. The first step is the derivation of the governing ODEs or PDEs, usually from a set of conservation laws that represent mathematical statements of the behavior of the physical system. For example, a code that simulates heat conduction in a stationary medium may be based on the following equations derived from the conservation of energy [9]:

$$\nabla \cdot k\nabla T + g = \rho C_p \frac{\partial T}{\partial t} \tag{1}$$

where ρ is the mass density, C_p is the specific heat, T is the temperature, k is the thermal conductivity (n×n matrix), and g is the heat generation rate.

Computer codes that solve Equation 1 will not all be the same because of a number of assumptions that can be made. For example, one code might restrict itself to isotropic heat conduction, in which the matrix k reduces to a scalar. Another code might consider only steady-state heat conduction in two dimensions while another code simulates transient heat conduction in three dimensions. A still more general code may consider nonlinear phenomena wherein the thermal conductivity is a function of temperature. Each of these codes would solve a slightly different set of governing equations.

The *interior* equation (1) does not, by itself, define a unique solution because it lacks boundary conditions and a statement of the domain on which the equation is to hold. Four general types of boundary conditions are usually applied to heat conduction phenomena:

$$\begin{aligned}
&Dirichlet && T = f(x, y, z, t) \\
&Neumann && \frac{\partial T}{\partial n} = f(x, y, z, t) \\
&Robin && k\nabla T + hT = f(x, y, z, t) \\
&Cooling + Radiation && -k\frac{\partial T}{\partial n} + q_{sup} = h(T - T_\infty) + \varepsilon\sigma(T^4 - T_r^4)
\end{aligned} \tag{2}$$

where h is the heat transfer coefficient, ε is the emissivity of the surface, and σ is the Stefan-Boltzmann constant, T_r is the effective temperature, and q_{sub} is the supplied heat flux. Other types of boundary conditions can also be posed. Some computer codes may be restricted to solving only a subset of these boundary conditions, others will be fully general. One also needs an initial condition,

$$T(x, y, z, t_0) = T_0(x, y, z) \tag{3}$$

We have previously stated that to verify software that solves PDE's by some ordered method, one must demonstrate that the governing equations are solved consistently. To make this statement

more rigorous, we must account for the fact that there are *auxiliary conditions*, namely, the boundary conditions, the initial conditions, and the domain on which the problem is formulated. PDE's that involve conservation statements on the domain are usually referred to as the *interior equations*. In this report, we will refer to the interior equations plus the auxiliary conditions as the *governing equations*. Thus, in our definition of code Verification, solving the equations correctly refers not only to the interior equation but to particular set (or sets) of boundary conditions, initial conditions, and domains permitted by the code. Code Verification entails demonstration that both the interior equations and boundary conditions are solved consistently. We will expand on this topic in Section 3.3

2.2 Discretization, Consistency, and Order-of-Accuracy

Discretization of the governing equations subdivides the domain of the problem into finite cells or elements. Rather than a continuum partial differential equation being solved for a solution in an infinite dimensional function space, the equations are solved on a finite dimensional subspace which approximates the continuum solution. The approximate solution, which satisfies the discretized equations, is not the same as the exact solution which satisfies the mathematical continuum equations. The difference between the two is called the *discretization error*. Discretization methods are *consistent* if the error goes to zero as the representative cell size h decreases to zero. The rate at which the error decreases to zero is called the order-of-accuracy. For example, a discretization method is said to be *second-order* accurate in space if the discretization error goes to zero as h^2. For brevity, we refer to a code containing a n^{th}-order accurate discretization method as a n^{th}-order accurate code. Terms in the governing equations may be discretized with different orders of accuracy. For example, it is not uncommon in an advection-diffusion equation for the advection term to be discretized to first-order accuracy while the diffusion term is second-order accurate. The over-all accuracy of such a code is the lowest order of all the approximations made (first-order in this example). A code may be first-order in time if the discretization error goes to zero as Δt goes to zero and second-order in space. Even though two codes may solve the same mathematical equation set, they can still differ in the method of discretization. For example, the equations can be discretized by various finite difference, finite volume, finite element, or boundary element methods. One code may be second-order while another is fourth-order spatially accurate.

Finally, with a discretization method chosen and some statement of the corresponding order-of-accuracy, one is ready to implement the discrete algorithm in a piece of software called *code*.

2.3 Software Quality Assurance (SQA)

We briefly discuss Software Quality Assurance because some of the methods of SQA are relevant to code Verification. SQA is a formal procedure developed to ensure that software systems are reliable and secure [10, 11, 12, 13, 14]. Current research in SQA is driven by high integrity systems such as control systems for aircraft and spacecraft, software for nuclear weapons and safety, and control systems for nuclear power reactors. The scientific software community has much to learn from SQA procedures developed for these systems.

The part of SQA that is most relevant to code Verification is the set of procedures developed for software testing. These procedures consist of three parts: static, dynamic, and formal testing. Static

testing is an important prerequisite to the Verification process. These tests are performed without running the code. Obviously, we need a code that compiles and is free of compilation errors. Next, the code is checked for consistency in the usage of the computer language. This step requires additional software such as commonly used Lint or Fortran checkers. These software tools are excellent in finding variables that are used but not initialized and also, in matching the argument list of a calling statement with the subroutine or the function argument list.

The static tests described above are not sufficient to identify all coding and conceptual mistakes. After completion of static testing one performs dynamic testing in which the code is run and checked for array indices that are out of bounds, for memory leakage, coverage issues and other problems that can be flagged by the compiler or other software. Code Verification can be viewed as an essential part of dynamic testing of engineering software. After the code has been Verified, remaining errors can be identified during formal analysis.

To this point we have been casual concerning our use of the word "error." To avoid confusion we must carefully distinguish between two relevant meanings of this word. In code testing, the word error generally refers to code mistakes (bugs) of the type already described in this Section. On the other hand, in code Verification we are concerned both with code mistakes and discretization error. Discretization error is the result of solving the continuum PDEs numerically and is not a code mistake. Discretization error is related to measures of the difference between the exact solution to the governing equations and the numerical solution to the discretized equations (we will discuss discretization error in detail in Section 4). For the remainder of this report we use the word "error" to refer to discretization error and the word "mistake" to refer to coding bugs or blunders.

2.4 A Taxonomy of Code Mistakes

For the purpose of code Verification, it is useful to define a particular taxonomy of code mistakes (see Figure 1). Mistakes detected by the static tests described in the previous Section we call *static mistakes*. Mistakes detected by running the code we call *dynamic mistakes.* Mistakes that remain after static and dynamic tests we call *formal mistakes*. An example of a formal mistake would be a line of code that calculates a quantity that is not used. Dynamic mistakes affect codes in several ways; they can affect code *convergence, efficiency, and order-of-accuracy.* A coding mistake that reduces the observed order-of-accuracy will be referred to as an Order-of-Accuracy Mistake (OAM). Order-of-accuracy mistakes can be further sub-divided into those which are severe enough to reduce the functional order-of-accuracy of the code to zeroth-order or below and those mistakes that reduce the order to a lesser degree. The former type of mistake can be referred to as a *consistency* mistake. With these definitions, we can now interpret the definition of code Verification (demonstration that the code is "solving the equations correctly") as being the process of identifying and eliminating OAM's. In dynamic testing, a code can occasionally diverge. This can happen for two basic reasons: (1) bad input or (2) a dynamic coding mistake. A coding mistake that causes the code to diverge we call a *divergence mistake*. *Efficiency mistakes* are mistakes which prevent the code from attaining its theoretical performance. Other dynamic coding mistakes also exist (for example, an incorrect output format). By definition, code Verification should catch all order-of-accuracy mistakes and may also catch some coding mistakes of the other types. To complete our taxonomy, *conceptual mistakes* are mental misconceptions involving the description of the algorithm that, when implemented, cause the code to not perform according to expectation. For example, a researcher devises a method for solving a non-linear equation which he believes to

be quadratically convergent, but code testing reveals that the method is, in fact, only linearly convergent.

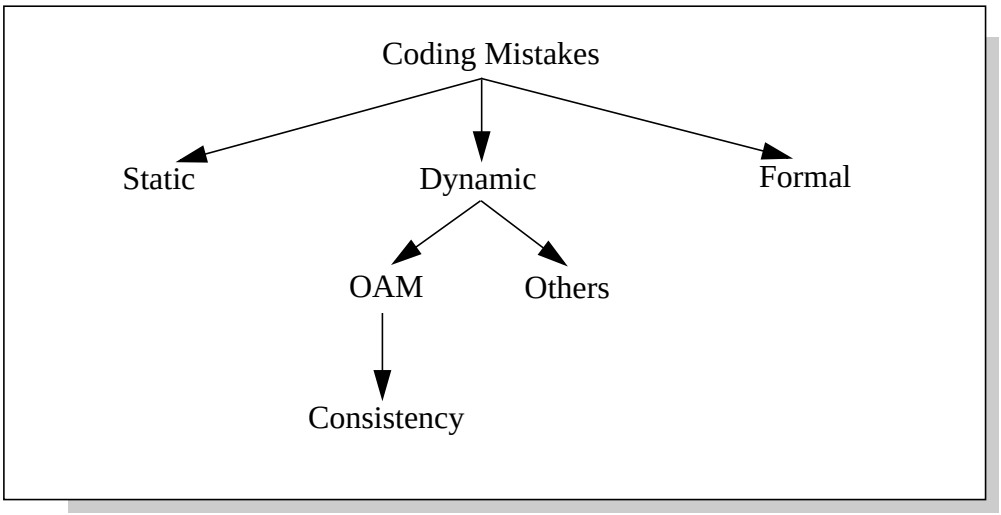

Figure 1. Hierarchical taxonomy of coding mistakes

2.5 Methods of Dynamic Code Testing

Dynamic tests related to code Verification are generally designed with two parts. The first part involves test selection and the second part test acceptance criteria. The following list provides commonly used dynamic testing approaches that have been applied by practitioners in the past. Of course, some tests are harder to pass than others. As we move from the top of the list to the bottom, we generally increase the difficulty of the test. In practice, one may chose more than a single testing approach. Only MES and MMS are appropriate for code Verification.

Dynamic Testing approaches

- ➢ Trend
- ➢ Symmetry
- ➢ Comparison
- ➢ Exact (MES)
- ➢ Manufactured solution (MMS)

The next list gives possible acceptance criteria which might be used for a given testing approach, in order of increasing rigor.

Acceptance Criteria

- ➢ Expert Judgement
- ➢ Percent Error
- ➢ Consistency
- ➢ Order-of-Accuracy

2.5.1 Trend Tests

In the Trend Method, a set of calculations (whose solution is not known) is performed, by varying input parameters. Whether or not the code passes the test is determined by *expert judgement*, in which the solution is examined for the right *trend*. For example, a heat conduction code may be run to a steady-state solution with several values of the specific heat and, if the time it takes to reach steady state decreases as the specific heat is decreased, then the expected trend has been reproduced. Also, the steady-state solution is expected to be smoothly varying so, if the numerical solution does not have this property, a coding mistake is suspected. The trouble with this approach is that codes can readily produce plausible solutions that are quantitatively incorrect. For example, although the trend towards the time to reach steady-state is correct, the true time to steady-state might be incorrect by a factor of two because of order-of-accuracy mistakes in the time derivative. The trend method sets a very low bar for a code to jump over and should be considered a minimal test that is best used during the code development stage, not the code Verification stage. Most unverified codes would pass the physical trend test.

2.5.2 Symmetry Tests

The Symmetry Method checks for solution symmetries which can be detected with no knowledge of the exact solution. We mention three cases. The first case is to set up a problem which, *a priori*, must have a spatially symmetric solution (perhaps due to the choice of boundary conditions). For example, the solution to a fully-developed channel flow problem can be symmetric. One inspects the solution to see if the code generates a symmetric answer. In the second case one checks for coordinate invariances such as translation and rotation. In this case the solution need not be symmetric. One calculates the solution to some problem and can then do the following tests (a) translate and rotate (say, by 90 degrees) the domain in physical space and (b) translate and rotate the coordinate systems. The solution should remain the same as in the original problem. In the third case, one performs a symmetric calculation using a 3-D code. The symmetry is constructed so that the results can be compared to a 2-D solution that is known analytically. The symmetry procedures provide a simple and effective way of testing the code for possible coding mistakes. This set of tests is most suitable for code development.

2.5.3 Comparison Tests

Another widely used approach to code testing is the Comparison Method in which one code is compared to an established code or set of codes that solve similar problems. A test case (usually realistic in practice) is devised and all the codes are run on the same test case. The usual acceptance criteria is based on a computation of the maximum difference in the solutions. In practice, if the percent difference between the solutions is within some tolerance, then the code is considered likely to be free of coding mistakes. The main advantage of this test is that it does not require an exact solution. Usually, calculations are performed on only a single grid. The procedure just outlined is incomplete because grid refinement is needed to determine whether the results from the codes are in the asymptotic range (this is seldom done in practice.)

The major difficulty with the Comparison Method centers around the fact that the codes usually are not identical, neither in the set of equations and input parameters that they solve nor the numerical method used. This often results in *apples* to *oranges* type comparisons wherein the inputs to the codes have to be *fudged* in some way in order to make any kind of comparison. Judgment of code correctness ends up based on ambiguous results. A related problem with the

Comparison Method is that, sometimes, no comparable code can be found. Another problem is that, in practice, comparison tests are usually run on a single physically realistic problem, therefore the comparison lacks generality. Reliability of the conclusions reached in the comparison test depend heavily on the confidence that can be placed in the established code. A successful comparison test does not rule out the possibility that both codes contain the same mistake.

2.5.4 Method of Exact Solutions (MES)

A widely-used method, known as the Method of Exact Solutions (MES), is more rigorous and objective than either the Trend or Comparison tests, In the MES code Verification procedure one first seeks exact solutions (or a benchmark) to the set of equations solved by the code. This is typically done by consulting the literature for published solutions. One can derive their own exact solution, for example, using mathematical methods such as the separation of variables, or integral transform techniques (Green's functions, Laplace Transforms, etc.).[1] The exact solution is a closed-form mathematical expression that gives values of the solution at all locations in space and time. For the heat conduction problem, one finds a solution to the partial differential equation corresponding to a set of material properties, initial conditions, and boundary conditions. Having this solution, the code is then run with corresponding inputs and the numerical (discrete) solution is compared to the exact solution. If the code fails to pass the acceptance criterion, then a coding mistake is suspected. If the code passes a comprehensive suite of such tests, the code is considered adequately Verified, i.e., the probability of a coding mistake is deemed small.

The MES procedure is capable of verifying codes according to the definition adopted in Section 1. However, as with all methods, there are limitations. Recall that in our definition of code Verification, one must demonstrate that the governing equations are solved correctly. The governing equations are usually rather general, involving nonlinearity, coupling between equations, complex boundary conditions, variable coefficients, and geometrically complex domains. A difficulty arises at this point because methods to find exact solutions to the governing mathematical equations usually require simplifying assumptions in order to solve the equations. For example, in the Laplace transform technique one assumes that the coefficients of the PDE are constants. If the computer code solves the heat conduction equation for heterogeneous materials having different thermal conductivities, then a Laplace transform solution will not test this code capability. Another common technique is to find exact solutions to one-dimensional problems because this is more easily accomplished. A one-dimensional solution is clearly inadequate to verify codes which simulate two- and three-dimensional heat conduction. Exact solutions often rely on the physical domain having a simple shape such as a rectangle or disk. Many modern computer codes can simulate physical phenomenon on very general shapes using finite element meshes or boundary-conforming structured grids. Such a capability is not adequately tested by comparing to exact solutions on simple domains because many of the terms in the equation become trivial. Exact solutions are often obtained by assuming that the domain is infinite or semi-infinite. Such domains don't fit within digital computers very well unless the code permits one to use a coordinate transformation.

To adequately verify a modern computer code, it is necessary to have solutions which test the fully general equations solved by the code. For example, there exist in the literature exact solutions to

1. See Appendix B for an example where an exact solution to the incompressible Navier-Stokes equations is given [9].

various equations from fluid dynamics [15]. In general, however, one can rarely find a single exact solution that solves the fully general set of governing equations. The MES work-around for this difficulty is to require a comprehensive suite of analytic solutions which, in total, comes closer to showing that the general governing equations are solved correctly. *The major limitation of the MES procedure is that it is difficult, if not impossible, to create a comprehensive suite of tests that adequately exercises the fully general set of governing equations.* Thus, code mistakes can still exist even after the MES procedure is performed.

A secondary limitation of MES is that exact solutions derived via Laplace Transforms, etc., can be very difficult to implement. Given an exact solution, one must compute its value at a number of points in space and time in order to compare it with the numerical solution produced by the computer code to be Verified. This is often very difficult to do because the exact solutions produced by the classical mathematical techniques often involve infinite sums of terms, non-trivial integrals, and special functions (such as Bessel). For an example of such a solution see Appendix A. To evaluate an exact solution containing an infinite series one encounters the problem of convergence and when to terminate the series. If an integral is involved, one must consider which numerical integration scheme to invoke and its accuracy. Sometimes the integrand in these exact expressions contains a singularity in the domain of integration which leads to a host of numerical difficulties. Evaluating such an integral numerically often requires consulting with an expert. As a general recommendation, developers of codes that implement analytic solutions to the governing equations should provide an assessment of the accuracy (how many significant figures) of the computed exact solution. Sometimes, this can be a challenge. A good example of this is an analytical solution for contaminant transport in a system of parallel fractures by Sudicky and Frind [16]. Van Gulick [17] has examined this analytical solution for accuracy and showed how difficult it is to obtain accurate solution to five significant figures; also, he showed in some cases the solution did not converge.

Another limitation that can arise with MES is that exact solutions may contain singularities. One cannot apply the order-of-accuracy criterion to solutions containing singularities. Even consistency can be difficult to demonstrate in such cases due to practical limitations on the number of grid points.

2.5.5 Method of Manufactured Solutions (MMS)

A more comprehensive (and often easier) alternative to the method of exact solution is the Method of Manufactured Solutions (MMS). This technique is the main subject of this report and is described in detail in Section 3.

2.5.6 Acceptance Criteria

In each of the testing approaches there must be a test acceptance (pass-fail) criterion. The four possible criteria that could be used in these tests were the expert judgement criterion (used in the Trend and Symmetry tests), the percent difference/error criterion (used mainly in the comparison method), the consistency criterion, and the order-of-accuracy criterion. Recall that a discretization of a PDE is consistent if, as the mesh size decreases to zero, so does the discretization error. If the mesh size is measured by the parameter h, then a consistent method will result in error that is proportional to h^p, where $p>0$. In code testing this means that the code should reduce the error to within machine roundoff with a sufficiently fine mesh. In practice this criterion, which requires

calculations of the solution on several mesh sizes, is seldom used. The most rigorous acceptance criterion is Verification of order-of-accuracy, in which one not only seeks to verify that the method is consistent, but also establishes the value of *p*, which is then compared to the theoretical order of the discretization method.

For each testing method one may choose among these four acceptance criteria. For example, MES could be associated with either the percent error, consistency, or the order-of-accuracy criteria. Thus there are at least three possible varieties of MES, with the order-of-accuracy criterion being the most rigorous. In the same way, the method of manufactured solutions (MMS) can be associated with one of these three acceptance criteria. In the following description of MMS we advocate that MMS be associated with the order-of-accuracy criterion, resulting in the most comprehensive and rigorous of all code Verification methods. Table 1 shows the possible associations between test methods and acceptance criteria. "Yes" signifies that the association is applicable. The Trend and Symmetry methods focus mainly on physical trends and do not involve mesh refinement, thus the last three acceptance criteria are not usually applied. The comparison method compares solutions from two or more codes. Typically, only the first two acceptance criteria are applied. Both MES and MMS use exact solutions so the last three acceptance criteria can theoretically be applied. In practice MES often uses the percent error criterion but can be made more rigorous by adopting the consistency or order-of-accuracy criteria. MMS can rigorously be applied using the order-of-accuracy criterion.

Table 1: Possible associations between code testing methods and acceptance criteria

		Acceptance Criteria			
		Expert Judgement	Percent Error	Consistency	Order-of-Accuracy
Testing Approaches	Trend	**Yes**	No	No	No
	Symmetry	**Yes**	No	No	No
	Comparison	**Yes**	**Yes**	No	No
	MES	No	**Yes**	**Yes**	**Yes**
	MMS	No	**Yes**	**Yes**	**Yes**

3 Method of Manufactured Solution (MMS)

A manufactured solution is an exact solution to some PDE or set of PDE's that has been constructed by solving the problem *backwards*. Suppose one is solving a differential equation of the form

$$\boldsymbol{D}\boldsymbol{u} = \boldsymbol{g} \tag{4}$$

where $\boldsymbol{D}$ is the differential operator, $\boldsymbol{u}$ is the *solution*, and $\boldsymbol{g}$ is a *source* term. In the method of exact solution (MES), one chooses the function $\boldsymbol{g}$ and then, using methods from classical applied mathematics, inverts the operator to solve for $\boldsymbol{u}$. In MMS, one first *manufactures* a solution $\boldsymbol{u}$ and then applies $\boldsymbol{D}$ to $\boldsymbol{u}$ to find $\boldsymbol{g}$ (a much simpler procedure). In the MMS code Verification *procedure*, one manufactures a solution to the fully general set of interior equations solved by the code that is to be Verified. The reason a solution to the general equations is wanted is to test all portions of the code.

Lets apply the manufactured solution technique to solve the heat conduction equation (1). The strength of this approach is that we can *create* a general solution that exercises all the terms and the coefficients of the interior equation without imposing special coordinate systems or particular boundary conditions.

The following is a manufactured solution and associated coefficient functions that we designed for Equation 1.

$$T(x,y,z,t) = T_0\left[1+\sin^2\left(\frac{x}{R}\right)\sin^2\left(\frac{2y}{R}\right)\sin^2\left(\frac{3z}{R}\right)\right] e^{t(t_0-t)/t_0} \tag{5}$$

$$k(x,y,z) = k_0\left(1+\frac{\sqrt{x^2+2y^2+3z^2}}{R}\right) \tag{6}$$

$$\rho(x,y,z) = \rho_0\left(1+\frac{\sqrt{3x^2+y^2+2z^2}}{R}\right) \tag{7}$$

$$C_p(x,y,z) = C_{p0}\left(1+\frac{\sqrt{2x^2+3y^2+z^2}}{R}\right) \tag{8}$$

where k_0, r_0, C_{p0}, t_0, T_0, and R are constants which determine the units and range of each function. The source term for the manufactured solution is computed from

$$g(x,y,z,t) = \rho C_p\frac{\partial T}{\partial t} - \nabla\cdot k\nabla T \tag{9}$$

With the help of the symbolic manipulation code Mathematica we substituted Equations 5-8 into Equation 9 and evaluated the source term *g*. The following is an expression for *g* directly from an output of Mathematica (the constant R has been replaced with Rad):

$$
\begin{aligned}
\text{Out[16]} = -\frac{1}{\text{Rad}^3}\Bigg(& e^{t-\frac{t^2}{t0}}\, u0 \Bigg(8\, k0 \left(\text{Rad} + \sqrt{x^2 + 2y^2 + 3z^2}\right) \cos\left[\frac{2y}{\text{Rad}}\right]^2 \sin\left[\frac{x}{\text{Rad}}\right]^2 \sin\left[\frac{3z}{\text{Rad}}\right]^2 + \\
& 2\, k0 \left(\text{Rad} + \sqrt{x^2 + 2y^2 + 3z^2}\right) \cos\left[\frac{x}{\text{Rad}}\right]^2 \sin\left[\frac{2y}{\text{Rad}}\right]^2 \sin\left[\frac{3z}{\text{Rad}}\right]^2 - \\
& 10\, k0 \left(\text{Rad} + \sqrt{x^2 + 2y^2 + 3z^2}\right) \sin\left[\frac{x}{\text{Rad}}\right]^2 \sin\left[\frac{2y}{\text{Rad}}\right]^2 \sin\left[\frac{3z}{\text{Rad}}\right]^2 + \\
& \frac{k0\, \text{Rad}\, x \sin\left[\frac{2x}{\text{Rad}}\right] \sin\left[\frac{2y}{\text{Rad}}\right]^2 \sin\left[\frac{3z}{\text{Rad}}\right]^2}{\sqrt{x^2 + 2y^2 + 3z^2}} + \frac{4\, k0\, \text{Rad}\, y \sin\left[\frac{x}{\text{Rad}}\right]^2 \sin\left[\frac{4y}{\text{Rad}}\right] \sin\left[\frac{3z}{\text{Rad}}\right]^2}{\sqrt{x^2 + 2y^2 + 3z^2}} \\
& \frac{1}{t0} \Bigg(cp0\, \text{Rad}\, rho0\, (2t - t0) \left(\text{Rad} + \sqrt{2x^2 + 3y^2 + z^2}\right) \left(\text{Rad} + \sqrt{3x^2 + y^2 + 2z^2}\right) \\
& \left(1 + \sin\left[\frac{x}{\text{Rad}}\right]^2 \sin\left[\frac{2y}{\text{Rad}}\right]^2 \sin\left[\frac{3z}{\text{Rad}}\right]^2\right)\Bigg)\Bigg)\Bigg)
\end{aligned}
$$

Usually, the source term created by the method of manufactured solutions is a *distributed source*, i.e., not a point source. This can give rise to a difficulty in the case that the code to be Verified does not allow distributed sources. A related difficulty is that the code may not contain a source term at all. These difficulties are discussed in Appendix B.

Note that the source term is complicated, which raises a question of how can one code this without making a mistake. The nice thing about Mathematica (and similar symbolic manipulation codes) is that it can create FORTRAN or C statements of the source term. From that one can create an auxiliary code to evaluate the source term at the required locations and time.

There are many other possible manufactured solutions to Equation 1 besides the one given.

The analytic solution to Equation 1 obtained by MMS is very general (being heterogeneous and non-steady) and exercises all the terms and coefficients in the governing equation. There is no difficulty computing the solution accurately since, by design, the solution consists of simple (primitive) functions that can be evaluated to virtually machine accuracy.

We have not, to this point, discussed how the domain, the initial condition, and boundary conditions are handled by MMS. These will be discussed in Section 3.3, but first we cover some guidelines for the construction of the solution to the interior equations.

3.1 Guidelines for Creating Manufactured Solutions

When constructing manufactured solutions, we recommend that the following guidelines be observed. The guidelines are needed to ensure that the numerical solution can be computed accurately and with little difficulty. It is usually not hard to create solutions within these guidelines.

First, manufactured solutions should be composed of smooth analytic functions like polynomials, trigonometric, or exponential functions so that the solution is conveniently computed (e.g., no infinite series). Solution smoothness is essential to ensure that the theoretical order-of-accuracy can be attained.

Second, the solution should be general enough that it exercises every term in the governing equation. For example, do not choose temperature T in Equation 5 to be independent of time.

Third, the solution should have a sufficient number of non-trivial derivatives. For example, if the code that solves the heat conduction equation is second-order in space, picking T in Equation 5 to be a linear function of time and space will not provide a sufficient test because second-order accurate convergence rates would be assured.

Fourth, solution derivatives should be bounded by a small constant. This ensures that the solution is not a strongly varying function of space and/or time. If this guideline is not met, then one may not be able to demonstrate the required asymptotic order-of-accuracy using practical grid sizes. Usually, the free constants that are part of the manufactured solution can be selected to meet this guideline. Finally, this guideline means that the solution should not contain singularities within the domain.

Fifth, the manufactured solution should not prevent the code from running successfully to completion during testing. Robustness issues are not a part of code Verification. For example, if the code assumes the solution to be positive, then make sure the manufactured solution is positive. Or, if time is expected to be positive, make sure the solution is defined for $t > 0$. Or, if the heat conduction code expects time units of seconds, do not give it a solution whose units are nano-seconds.

Sixth, the solution should be defined on a connected subset of two- or three-dimensional space. This allows flexibility in selecting the domain of the PDE (see Section 3.3.1)

Seventh, the solution should be constructed in a manner such that the differential operators in the PDE's make sense. For example, in the heat conduction equation, the flux is required to be differentiable. Therefore, if one desired to test the code for the case of discontinuous thermal conductivity, then the manufactured solution for temperature must be constructed in such a way that the flux is differentiable (see [7] for an example of where this was done).

The reader should realize that there is no requirement for physical realism in the manufactured solution. We maintain that such a requirement is not necessary and, in fact, is detrimental because it is often more difficult to construct a general test problem. The code cannot make a distinction between a physically realistic problem and a non-physical one since it is only solving a set of equations.

3.2 Guidelines for Construction of the PDE Coefficients

In constructing a manufactured solution, we are free to choose the PDE coefficient functions, with mild restrictions similar to those imposed upon the solution function.

First, coefficient functions should be composed of analytic functions like polynomials, trigonometric, or exponential functions so that they are conveniently computed (e.g., no infinite series).

Second, the coefficient functions should be non-trivial and fully general. In the heat conduction equation, for example, one should not choose the conductivity tensor to be a single constant scalar. Such a choice would fail to exercise the full code capabilities. To test the full capabilities one should choose the conductivity to be a full tensor. Well-designed codes will permit one to have spatial discontinuities in the conductivity functions and so these should be included as well (see [1] or [7] for an example of where this was done).

Third, the coefficient functions (which usually represent material properties) should be somewhat physically reasonable. For example, although one can construct solutions to the PDE in which the specific heat is negative, this violates conservation of energy and is likely to cause the code to fail over some robustness issue. Another example: the conductivity tensor is mathematically required to be symmetric, positive definite in order for the equation to be of elliptic type. If this condition is violated in the construction of the manufactured solution, one again would likely run afoul of some numerical robustness issue present in the code (e.g., if an iterative solver were used, the symmetric, positive-definite assumption is likely made in the design of the iterative scheme). One does not need to go over-board on physical realism, however; even though there is no material whose conductivity varies like a Sine function, this is often a useful choice.

Fourth, the coefficient functions need to be sufficiently differentiable so that the differential operator make sense. For example, one should not manufacture coefficient functions which contain a singularity within the domain.

Fifth, the coefficient functions should be within the code's range of applicability. If the code expects heat conductivities to be no smaller than that of cork, then don't give it a problem with near-zero conductivity. Again, the object in Verification is not to test code robustness.

3.3 Auxiliary Conditions

To this point we have shown how to manufacture a general solution to the *interior* equations and given guidelines for constructing the solution and the PDE coefficients. As noted in Section 2.1, the governing equations are not complete until the auxiliary conditions (initial condition, boundary conditions, and problem domain) are determined. *An essential difference between the MES and MMS code Verification procedures lies in the treatment of the auxiliary conditions.* With MES, the solution to the governing equations is derived by considering the interior equations and auxiliary conditions simultaneously. In MMS, the auxiliary conditions can often be considered after the solution to the interior equation has been manufactured. This flexibility is one of the most attractive features of MMS. In this Section we discuss how the auxiliary conditions are determined for code Verification by MMS.

To illustrate this difference between the two code Verification procedures, we begin with the *initial condition*, which is relatively easy to understand. In MES, initial conditions are given and one attempts to find an exact solution to the equation that satisfies the governing equations (including the initial condition). In the MMS procedure the initial condition poses no difficulty because the solution to the interior equation is already known. Once the manufactured solution $u(x,y,z,t)$ has been constructed, the initial condition u_0 is simply found by evaluating the manufactured solution u at $t=t_0$, i.e., $u_0(x,y,z) = u(x,y,z,t_0)$. For example, in our manufactured solution to the heat conduction equation, Equation 5 yields

$$T(x, y, z, t_0) = T_0\left[1 + \sin^2\left(\frac{x}{R}\right)\sin^2\left(\frac{2y}{R}\right)\sin^2\left(\frac{3z}{R}\right)\right] \tag{10}$$

for the initial condition. In the MMS code Verification procedure, this function is evaluated at discrete locations dictated by the mesh and the values of the function are given to the code as input.[1] Because the manufactured solution solves the fully general equations, the input initial condition derived from the manufactured solution is also general (e.g., spatially varying).

To achieve flexibility, engineering codes usually solve more than one set of auxiliary conditions. For example, the code may provide options for various boundary condition types (e.g., slip, no-slip, inflow, outflow, symmetry, and periodic). The code is usually flexible concerning the location on the domain boundary where each type is to be imposed. A number of code Verification issues result from this flexibility which are discussed in the subsections to follow.

3.3.1 Choosing the Problem Domain

Because the manufactured solution and the PDE coefficient functions are defined on some connected subset of 2-D or 3-D space, one has considerable freedom in the selection of the problem domain[2]. Limitations on the domain type may be imposed by the particular code being Verified. For example, the code may only allow rectangular domains. Because we wish to test the code at its highest level of generality, we choose the domain as general as possible. If the most general domain the code is capable of handling is a rectangle of sides a and b, then do not choose a square domain, choose instead the rectangle. If the code can handle general, simply-connected 2-D regions, do not use a rectangle for the domain. If the code handles non-simply connected domains, and this option is to be Verified, then choose the domain accordingly. If the code is 3-D, use a 3-D domain. If the most general domain type is used then one is more likely to find coding mistakes than if only special cases are tested. An attractive feature of the MMS procedure is that the domain can often be selected after construction of a manufactured solution.

With the MMS procedure, the process of selecting the problem domain is often, but not always, independent of constructing the boundary condition input to the code. Some exceptions are considered in the next subsection, on boundary conditions.

1. Strictly speaking, we recommend that one *not* use the initial condition derived from the exact solution. Although, this would minimize the number of iterations needed to converge the solution, the danger of this approach is that it can hide coding mistakes (see Section E.4 in Appendix E for an example). We recommend that one use the initial condition derived from the manufactured solution multiplied by some constant not too close to one.
2. For example, Equation 5 shows that any subset of R^3 is a potential domain.

3.3.2 Boundary Conditions

Treatment of boundary conditions is one of the most difficult topics in code Verification. Boundary conditions generally possess three important attributes: *type, location, and value*. As mentioned earlier, there are many different *types* of boundary conditions such as Dirichlet, Neumann, and Mixed. The second attribute of boundary conditions is *location*, i.e., the portion of the domain boundary on which each boundary condition type is imposed. For example, if the domain is a rectangle, perhaps the top and bottom boundaries are flux while the left and right are Dirichlet. The third attribute is *value*, i.e., the numerical value imposed by the boundary condition at a given location. For example, the flux at a point may be zero (a homogeneous flux condition) or non-zero (an in homogeneous flux condition). Either way, the numerical value is one of the required code inputs for this boundary condition type. Another important observation concerning boundary conditions is that one needs boundary conditions for each dependent variable in the PDE.

Assume for the moment that both the type and location of the boundary condition has been given. Then, in the MMS procedure the value attribute can be computed from the manufactured solution.[1] A very attractive feature of MMS, then, is that boundary condition type and location can often be selected after construction of the manufactured solution. We illustrate this point for various boundary condition types.

Dirichlet Boundary Conditions. If a Dirichlet condition is imposed on a given dependent variable for some portion of the domain boundary, then the value attribute is easily computed by evaluating the manufactured solution to the interior equations at the relevant points in space dictated by the location attribute and the grid on the boundary. The calculated values form part of the code input, along with the boundary condition type and location. If the manufactured solution is time-dependent, then the value attribute must be calculated for each time step.

In the heat conduction example, the *value* of the Dirichlet condition on the boundary $x=L_x$ is

$$T(L_x, y, z, t) \;=\; T_0\left[1 + \sin^2\left(\frac{L_x}{R}\right)\sin^2\left(\frac{2y}{R}\right)\sin^2\left(\frac{3z}{R}\right)\right]\; e^{t\,(t_0 - t)/t_0} \tag{11}$$

Note that the boundary condition is time-dependent, as well as spatially dependent.

Neumann Boundary Conditions. Also known as flux boundary conditions, these take the form

$$K\nabla T \cdot \hat{n} \;=\; q \tag{12}$$

The value attribute for this boundary condition is the numerical values of the scalar q at each point in time and space. These values can be calculated from the manufactured solution by analytically calculating its gradient (perhaps with a symbolic manipulation code) and evaluating the analytic expression at the required points in space and time. The results are used as code input.

1. For this discussion it is assumed that, like the interior equations, boundary conditions are imposed as differential relationships between the dependent variables.

Cooling and Radiation Boundary Condition The fourth boundary condition in Equation 2 can be applied on the $x=L_x$ boundary by re-writing the condition as

$$k\frac{\partial T}{\partial n} + hT + \varepsilon\sigma T^4 - hT_\infty - \varepsilon\sigma T_r^{\,4} = q_{sup} \tag{13}$$

The left hand side of Equation 13 can be evaluated using the manufactured solution and values of k, h, ε, σ, T_∞, and T_r. Then, the result is provided to the code as an input for q_{sup} which in this case, is time and spatially varying. If the code input only allows q_{sup} to be constant, then a code modification is required. If one cannot modify the code, one may be forced to leave this particular code option unverified. This boundary condition provides an example which shows that code modification may be necessary to verify some boundary condition options by the MMS procedure.

Free Slip Boundary Conditions. This condition is a Dirichlet boundary condition. The component of velocity normal to the boundary is zero (the value attribute is zero); the component of the velocity parallel to the boundary is not prescribed. This boundary condition type requires that the manufactured solution have zero normal velocity at the given set of locations at which the condition is imposed. In this case the domain is not entirely independent of the boundary conditions. Never-the-less, it is quite possible to construct manufactured solutions adhering to this boundary condition. The trick to satisfying this boundary condition is to manufacture a solution to the interior equations for which the normal velocity equals zero on some curve. This curve then determines the part of the domain boundary on which the free-slip condition will be applied. A similar case can be found in [7], in which a zero-flux condition was enforced along a boundary curve.

Free-Surface Boundary Condition A manufactured solution for porous media flow which involves a free-surface boundary condition is found in [19].

In principle, MMS should be applicable to other boundary condition types (such as No-Slip, Periodic, Symmetric, Inflow, and Outflow) because most are Dirichlet, Neumann or Mixed boundary condition types in disguise. Full assessment of the difficulties involved in each of these cases awaits further development of the method.

To this point we have shown how to manufacture solutions to the interior equations and shown how the manufactured solution is used to determine boundary condition values. The domain and boundary condition types are determined by the code capabilities. Construction of the manufactured solution is only part of the MMS code Verification procedure. The full sequence of steps in the procedure is discussed in Section 5, but first we describe some issues related to evaluation of discretization error and order-of-accuracy.

4 Evaluation of Discretization Error and Order-of-Accuracy

4.1 Discretization Error

The numerical solution consists of values of the dependent variables on some set of discrete locations determined by the grid and discrete time levels. To compute the discretization error, several measures are possible. Let x be a point in R^n and dx the local volume. To compare functions u and v on R^n the L_2 norm of u-v is

$$\begin{aligned} \|u-v\|_2 &= \sqrt{\int (u-v)^2 dx} \\ &= \sqrt{\int (u-v)^2 J d\xi} \end{aligned} \tag{14}$$

where J is the Jacobian of the local transformation and $d\xi$ is the local volume of the logical space. By analogy, for discrete functions U and V the l_2 norm of U-V is

$$|U-V|_2 = \sqrt{\sum_n (U_n - V_n)^2 \alpha_n} \tag{15}$$

where α_n is some local volume measure and n is the index of the discrete solution location.

Because the manufactured solution is defined on the continuum, one can evaluate the exact (manufactured) solution at the same locations in time and space as the discrete solution. The *local* discretization error at point n of the grid is given by u_n - U_n, where u_n is the manufactured solution evaluated at x_n, y_n, z_n and U_n is the discrete solution. The normalized *global* error is defined by:

$$e_2 = \sqrt{\frac{\sum_n (u_n - U_n)^2 \alpha_n}{\sum_n \alpha_n}} \tag{16}$$

If the local volume measure is constant (e.g., as in a uniform grid), the normalized global error (sometimes referred to as the RMS error) reduces to

$$e_2 = \sqrt{\frac{1}{N}\sum_n (u_n - U_n)^2} \tag{17}$$

From Equation 16 one sees that if $u_n - U_n = \vartheta(h^p)$, then the normalized global error is $\vartheta(h^p)$. This fact enables one to ignore non-uniform grid spacing when calculating the discretization error for the purpose of code Verification (recall that one only needs to show consistency or verify the theoretical order-of-accuracy; the actual magnitude of the error is irrelevant). Thus, either Equation 16 or Equation 17 may be used in code Verification exercises.

The infinity norm is another useful norm of the global error, which is defined by

$$e_\infty = \max_n |u_n - U_n| \tag{18}$$

In general, both error measures should include points on the boundary.

Often engineering software computes not only the solution but certain derived variables such as flux or aerodynamic coefficients (lift, drag, moment). For thorough Verification of the code, one should also compute the error in these derived variables if they are provided as code output. For example, Verification of the calculated flux (which involves the gradient of u), can use the following global error measure

$$e_2 = \sqrt{\frac{1}{N}\sum_n (u'_n - U'_n)^2} \tag{19}$$

The gradient of the exact solution U'_n is easily calculated (perhaps with Mathematica) from the analytic expression for the exact solution. u'_n is obtained from the code output. It should be noted that the theoretical order-of-accuracy of calculated fluxes may be less than the theoretical order-of-accuracy of the interior equations.

An important point is that MMS procedure is not limited to any *particular* error measurement. For example, one can use either the RMS the maximum norm, or both.

4.2 Determination of Observed Order-of-Accuracy

Given that we can obtain some measure of the global error on a grid, we then run the code on a series of different mesh sizes h (at least 2) and compute the error for each mesh. From this information, the observed order-of-accuracy can be estimated and compared to the theoretical order-of-accuracy.

For example, if the discrete solution is spatially second-order accurate and in the asymptotic regime, the error (in the RMS or infinity norms) will decrease by a factor of 4 for every halving of the grid cell size. Often the spatial error is separated out from the temporal error.

The definition of the theoretical order-of-accuracy is related to the discretization error and is a function of h, where h is the grid spacing:

$$E = E(h) \tag{20}$$

For consistent methods, the discretization error E in the solution is proportional to h^p where $p>0$ is the theoretical order of the method

$$E = Ch^p + H.O.T \tag{21}$$

where C is a constant independent of h and $H.O.T$ refers to higher order terms. Because we have considerable flexibility in constructing the manufactured solution, it is not difficult to ensure that

it has sufficient derivatives so the Taylor series for the manufactured solution exists. Therefore, this description of error applies to every consistent discretization method, such as finite difference method (FDM), finite volume methods (FVM), and finite element methods (FEM). It is important that the numerical procedure be consistent, which means the continuum equations are recovered as *h* goes to zero.

To estimate the order-of-accuracy of the code being Verified we rely on the assumption that as *h* decreases to zero, the first term in Equation 21 dominates. The global errors, E_{grid1} and E_{grid2}, are obtained from the numerical solutions to the Verification test problems. If *grid1* is of size *h* and *grid2* is of size *h/r*, where *r* is the refinement ratio, the error has the form:

$$E_{grid1} \approx Ch^p \tag{22}$$

$$E_{grid2} \approx C\left(\frac{h}{r}\right)^p \tag{23}$$

The ratio of the errors is given by

$$\frac{E_{grid1}}{E_{grid2}} \approx \left(\frac{Ch^p}{Ch^p}\right)r^p = r^p \tag{24}$$

Thus, the observed order *p* is computed as

$$p \approx \log\left(\frac{E_{grid1}}{E_{grid2}}\right) / \log(r) \tag{25}$$

Systematic grid refinement with a constant grid refinement ratio *r* produces a sequence of observed orders-of-accuracy. The trend in the sequence is then compared to the theoretical order-of-accuracy of the discretization method. Typically one should not expect to recover the theoretical order-of- accuracy to more than two or three significant figures with this procedure because in practice reducing *h* to zero requires high precision and large numbers of elements which, in turn, result in excessive memory requirements and run-times.

To compute the observed time-order-of-accuracy one can use the same formulation presented above by replacing the grid size *h* with the time step size Δt. Thus, equation 25 is applicable to both time and space discretization.

5 Summary of the MMS Code Verification Procedure

In this Section we summarize the MMS code Verification procedure (idealized in Figure 2 on page 36).

Step 1. *Determine Governing Equations*. To begin, one must determine the system of governing equations solved by the code. The users manual is a good place to start but, if the documentation is inadequate, one may try consulting the code developer. Clearly, the code cannot be Verified if one does not know *precisely* what equations are solved. It is not enough to know that, for example, the code solves the Navier-Stokes equations. One must know the exact equations (e.g., compressible or incompressible). In particular, for MMS we must know the most general set of equations that are solved (e.g., is the Prandtl number permitted to be spatially variable?) Difficulties in determination of the governing equations arises frequently with commercial software, where some information is regarded as proprietary. *If the governing equations cannot be determined, we would question the validity of using the code*.

A related issue is that to verify the code by MMS procedure one should know the theoretical order-of-accuracy of the discretization method. In some cases, this may be difficult to ascertain from the documentation (or even the code author). If this cannot be determined, we recommend changing the acceptance criteria from order-of-accuracy to the consistency criterion, which any code with a consistent algorithm must obey.

Step 2. *Design Suite of Coverage Tests.* As noted in Section 3.3, to achieve flexibility, engineering codes usually solve more than one set of auxiliary conditions. These manifest themselves as code input options such as, for example, boundary condition type switches. In addition, other options may exist. For example, the code input may contain a switch to determine which solver is used on the linear system of equations. To ensure that all code options relevant to code Verification are tested, one must design a suite of coverage tests. Fortunately, this is not as daunting as it may seem at first. If a code has N options, the number of coverage tests needed to verify the code is determined by the number of *mutually exclusive* options, i.e., there is no combinatorial explosion of tests to run. For example, suppose a code has two solver options and three constitutive relationship (CR) options. Then only three coverage tests are needed to check all options. Test (1): Solver 1 with CR1, Test (2): Solver 2 with CR2,Test (3): Solver 1 or 2 with CR3. One does not need to perform a test involving the combination, for example, Solver 1 and CR2 because Test 1 will ascertain whether of not Solver 1 is working correctly, while Test 2 will ascertain whether or not CR2 is working correctly.

The case of boundary condition options deserved special consideration. Suppose a code has three options for different boundary condition types. Clearly each one must be tested, so the question is, what is the *minimal* number of tests that need be performed? This question is not easily answered in general due to the wide variety of specialized boundary conditions types. However, some situations are straightforward. Consider the case of a code having N boundary condition types. Let the boundary be divided into N non-intersecting subsets whose union is the original boundary. To fully test the code one can apply each of the N boundary conditions to each of the subsets. This results in N coverage tests for the boundary conditions.

A drawback to performing the minimal test suite is that Step 9 (searching for coding mistakes) can be made more difficult than a more systematic test suite which does not attempt to minimize the number of tests. By a process of elimination, options can be tested one at a time. In the example of a code having two solvers and three constitutive relations, one could perform the following tests: (1) Solver 1 with CR1, (2) Solver1 with CR2, (3) Solver 1 with CR3, and (4) Solver 2 with CR1. The most difficult test is the first because if coding mistakes are suspected, one must search in both the solver and the routine for the constitutive relation (as well as the portion of the code dealing with the interior equations). However, having passed that test, then if a coding mistake is suspected in Test 2, one need only search the routine for the second constitutive relationship. Clearly, designing the test suite is somewhat of an art form. It is important to invest a sufficient amount of time in Step 2 to save time during later phases of testing.

Note that the suite of coverage tests should include tests for the calculation of solution functionals such as flux and aerodynamic coefficients.

We recommend that, when possible, the first coverage test to be performed use Dirichlet boundary conditions on the entire boundary. If one uses the exact solution to compute the input Dirichlet boundary conditions, the error on the boundary will be zero if there is no coding mistake in the Dirichlet boundary option. This can be quickly checked by examining the local error in the solution. If the error on the boundary is indeed zero, the remaining error, if any, must arise from the interior equations. This procedure tests the Dirichlet boundary condition option and, more significantly, permits Verification of the interior equation independently of the boundary conditions. When subsequent coverage tests are run, any coding mistakes that remain must reside in the auxiliary conditions. The Verification procedure can thus be simplified by process of elimination. If this procedure is not followed, then searching for coding mistakes (Step 9) is less straightforward.

We provide an example to illustrate some of the subtleties that can arise when designing a test. Consider the case of a code having both Dirichlet and Neumann boundary conditions. Clearly two coverage tests are needed but, because one cannot usually run a code with Neumann conditions on all of the boundary at once, one must subdivide the boundary into two subdomains. Each subdomain will be assigned one of the two boundary conditions. Two tests result by swapping the boundary conditions of the subdomains.

An additional coverage test will be needed for any code which can perform transient calculations. To check the time order-of-accuracy of such a code there are two approaches: the first approach uses a fixed grid for which the spatial discretization error is nearly reduced to machine roundoff and then refine the time step for the grid convergence test; the second approach refines both the grid and time-step together. Both approaches should provide the same order-of-accuracy for time. We prefer the first approach because it is easier to isolate coding mistakes.[1] The additional coverage test will consist of a series of time-step refinements using the fine grid to compute the observed time-order-of-accuracy.

1. The first approach requires that one embed a switch in the manufactured solution to permit easy transition between steady and transient solutions.

As one tests the various boundary condition options within a code, it is possible that one could create an ill-posed problem. In this case, the code may fail to converge and one would be forced to redesign the test problem. Re-designing to avoid ill-posedness is usually not a major difficulty since one has a great deal of flexibility concerning the combinations of type and location of boundary conditions used.

Step 3. *Construct Manufactured Solution*. Once the governing equations have been determined and a particular coverage test has been selected, the next step is to construct a general manufactured solution. This step was described in Section 3. The main objective is to construct the most general solution and inputs permitted by the code that meet the guidelines given in Sections 3.1 and 3.2. This issue was discussed in detail in Section 3.

Step 4. *Perform the Test.* First, code inputs are calculated from the manufactured solution. Second the code is run and the output is used to calculate the global error. Third, calculate the observed order-of-accuracy using Equation 25.

Step 5. Grid Refinement. Grid refinement is performed to obtain errors on a sequence of grids. Because we want to test the code at its most general level, we should generate the most general grid the code is capable of using. If the code runs boundary-fitted coordinates, then do not design a test problem using Cartesian coordinates. If the code runs stretched tensor product grids, do not design the test problem with a uniform mesh. If the code allows non-uniform time-steps, make sure the test problem uses that capability. Because the manufactured solution is a closed-form expression defined everywhere on the problem domain, we can evaluate it at whatever grid locations are required to compute the discretization error.

A few words are necessary concerning the grid refinement procedure. For finite difference applications (and some finite volume methods) one should *always use smooth grids*. The primary reason is that if a non-smooth grid is used, one may decrease the observed order-of-accuracy. Recall that the grid metrics appear in the governing equations whenever a non-uniform grid is used. For example, under a transformation $x(\xi)$, $df/dx = (d\xi/dx)(df/d\xi)$. If both $df/d\xi$ and $d\xi/dx$ are discretized with second-order accuracy, then the theoretical order-of-accuracy of df/dx is second-order - *assuming that the grid is smooth*. If a non-smooth grid is used, then the observed order-of-accuracy of df/dx will be first-order. If one were verifying a code using a non-smooth grid, then the observed order-of-accuracy may not match the theoretical order and one would falsely conclude the code contained a coding mistake. A practical example of the importance of this observation is that of 1-D grid refinement by bisection of the physical grid cells. The resulting mapping underlying the coarse grid can be represented by a piecewise linear function. Thus, the underlying mapping is not smooth and, as a result, an order-of-accuracy in the grid metrics is lost.

Step 6. Compare Order-of-Accuracy. Compare the observed order-of-accuracy to the theoretical order. The observed order-of-accuracy is calculated from Equation 25 for each consecutive pair of grids. From this sequence of estimates, determine the order-of-accuracy of the code. This step is a branch point in the procedure. If the observed order-of-accuracy is less than the theoretical order then proceed to Step 7: Troubleshoot the Test Implementation. If the observed order-of-accuracy is equal to or exceeds the theoretical order-of-accuracy, proceed to Step 10: Is there another coverage test?

Step 7. *Troubleshoot Test Implementation.* If the theoretical order-of-accuracy is not achieved, one should consider the following three possibilities.

The *first* possibility is that there may have been a mistake in the test formulation or setup. Such as, a mistake in the codes' input deck, a mistake in the derivation of the manufactured solution, the associated coefficient functions, or the source term. One may also have a mistake in the software that calculates the source term or boundary condition input values.

The *second* possibility is that there may be a mistake in the assessment of the results. The software that calculates the exact solution at the given spatial and temporal locations could be wrong or perhaps the software that calculates the error norms and observed order-of-accuracy may have mistakes in them. It is also possible that one has not yet reached the asymptotic range, i.e., a fine enough mesh has not been employed. One should also consider that perhaps the theoretical order-of-accuracy of the numerical method is not what one thought it was.

The *third* possibility is that there may be incomplete iterative convergence. If the code uses an iterative solution method, either due to use of an iterative solver or by linearization of a non-linear PDE, then one must be sure that the iterative stopping criterion has been tightly set so that the true solution to the discrete equations has been obtained.

Step 8. *Implementation Problem Found?* If the three possibilities considered in Step 7 revealed a problem in the test implementation, then one fixes the problem and repeats the test (Step 4). If no problems in the implementation were found, then one proceeds to Step 9.

Step 9. *Search for Order-of-Accuracy Mistakes.* If all test implementation problems have been eliminated and still the expected order-of-accuracy has not been obtained, then one must seriously entertain the possibility that there is an order-of-accuracy mistake (OAM) in the code. Disagreement between the observed and theoretical order-of-accuracy indicates that there is an OAM, but it will not directly reveal the mistake.[1] Because of the generality of the manufactured solution, the mistake could reside anywhere in the code. However, it is possible to narrow down the location of the mistake by performing a series of less general tests. For example, in a heat conduction code one might revise the manufactured solution to be independent of time and repeat the mesh refinement study. If the theoretical order-of-accuracy is not obtained, then an OAM resides in that part of the code which deals with the spatial terms in the equation. Similarly, if the manufactured solution is restricted to spatially constant heat conductivity and the expected order-of-accuracy is obtained, then one can say that an OAM likely resides in that part of the code that deals with the heat conductivity arrays. A similar approach can be used by restricting the generality of the boundary conditions or domain shape to systematically isolate where the OAM lies. If the general manufactured solution is built cleverly, the special case solutions can be quickly obtained by simply setting certain parameters in the solution to zero. We note also that, if an OAM is found, one should not assume that it is the only OAM. The mistake should be fixed and the test repeated to determine if additional mistakes remain (Step 4).

1. It will become clearer to the reader how to search for OAM's in Section 7.

Step 10. *Another Coverage Test Needed?* Step 10 is reached if, in Step 6, the observed order-of-accuracy agreed with the theoretical order. In this situation, the current coverage test is completed and one proceeds to Step 11 if there is another coverage test, otherwise one proceeds to Step 12.

Step 11. *Do we need to return to Step 3?* One can sometimes use the same manufactured solution for different coverage tests. For example, testing of different solver options can clearly be done with the same manufactured solution. If a code has Dirichlet, Neumann, and Mixed boundary condition types, the same manufactured solution can be used because, as explained in Section 3.3, the solution to the interior equation and its derivatives are known on the boundary. Both steady and unsteady-state options can be tested with one solution by building into the manufactured solution a constant that multiples the time variable. Setting this constant to zero converts the unsteady solution to a steady solution. In some cases the code options can be build directly into the manufactured solution by including switches. Options for different viscosity models can be treated in this manner. Options which change the number of governing equations generally require more than one manufactured solution.If another manufactured solution is needed, proceed to Step 3, otherwise go to Step 4.

Step 12. *Code Verified*. This point is reached if all the coverage tests have been performed and all order-of-accuracy mistakes have been eliminated. *By definition, the code is Verified.*

Once the code is Verified, the issue of whether or not the equations are being solved correctly is closed until a code modification is made that potentially introduces OAM's. It should be a simple matter to re-verify the code if the initial Verification procedure and results were documented and any auxiliary codes (such as those which evaluate the source term and compute the error) were archived. In some cases a new manufactured solution may be required, for example, if an additional equation has been added to the governing equations.

Some codes have a variable order-of-accuracy. For example, shock capturing codes with limiters generally reduce the local order-of-accuracy in the vicinity of shocks To apply the MMS procedure to this situation, first manufacture a smooth solution to the governing equations. This solution will not exercise the limiters, i.e., the order-of-accuracy will not be reduced. The smooth solution can be used to verify all portions of the code except the part dealing with the limiters. To verify the limiters, one may then create an additional coverage test involving a non-smooth manufactured solution which exercises the limiters. The observed discretization error should then be greater than or equal to the lower bound on the theoretical order-of-accuracy of the limiter. Since the rest of the code has been Verified by process of elimination, Step 9 will then consist of searching the portion of the code dealing with limiters for coding mistakes.

It may be necessary to modify the testing procedure slightly if a code contains a constitutive relation involving a "look-up" table. The only way to establish the correctness of a "look-up" table is to compare the entries in the table to the reference by which it was constructed. The presence of a "look-up" table does not prevent the remainder of the code from being Verified by the MMS procedure. If a "look-up" table is the only constitutive relation provided by the code, one must modify the code to by-pass the table, adding, for example, a constant constitutive relation.

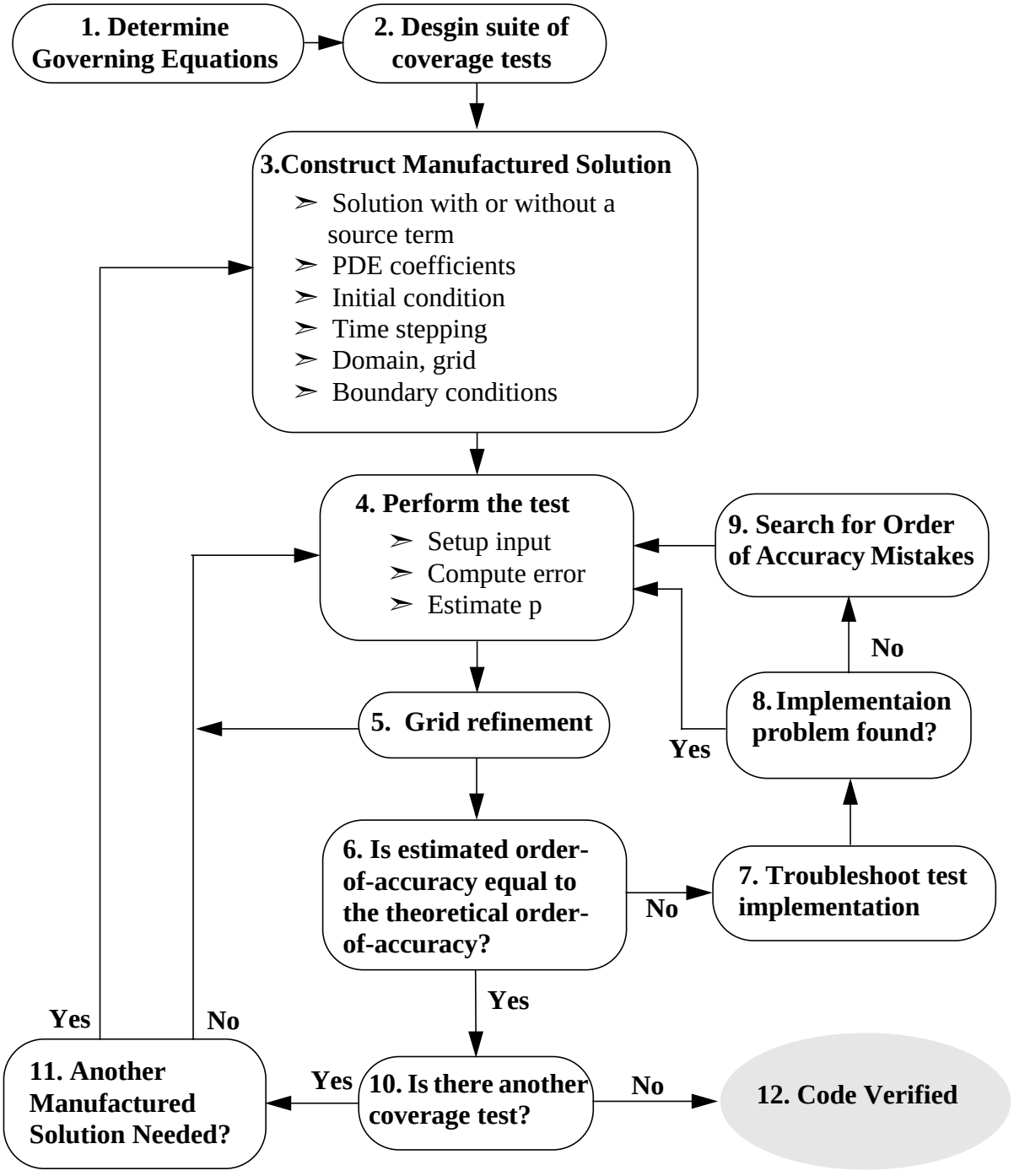

Figure 2. Flow chart for Code Verification by the method of manufactured solutions (MMS).

6 Application of Manufactured Solutions in Code Verification

In the following subsections and Appendix C, we verify four different codes, that were written specifically for this study to demonstrate how the MMS procedure is applied to non-trivial equations sets in fluid dynamics. The codes were written using a variety of different numerical methods to illustrate the general applicability of the MMS procedure. The codes Verified in this Section are not production codes but serve to illustrate the procedure. Additional Verification issues not previously mentioned will present themselves in this Section.

Three different sets of equations are solved by these codes: 2-D Burgers equation, 2-D laminar incompressible Navier-Stokes equations, and 2-D laminar compressible Navier-Stokes equations. The following is a list of options that are available within each code:

- ➢ *Time dependent 2-D Burgers equation (Code 1)*
  - Cartesian coordinates, collocated variables
    - Dirichlet boundary condition
    - Neumann boundary condition
    - Artificial dissipation
  - Curvilinear coordinates, collocated variables (Code 2)
    - Dirichlet boundary condition
    - Time dependent Dirichlet boundary condition
- ➢ *Time Dependent Laminar 2-D incompressible Navier-Stokes Equations (Code 3)*
  - Cartesian coordinates, staggered mesh
    - Dirichlet boundary condition
- ➢ *Time Dependent Laminar 2-D compressible Navier-Stokes Equations (Code 4)*
  - Cartesian coordinates, collocated variables
    - Dirichlet boundary condition

The 2-D Burgers equation provides a simple demonstration of the MMS procedure. We have two codes with different numerical schemes that solve the unsteady Burgers equation, one in Cartesian and the other in the curvilinear coordinates. Both codes use collocated variables. Two types of boundary conditions are implemented, Dirichlet and Neumann. In the curvilinear coordinate case, we verify the steady and the unsteady options in the code. Also, we investigate the influence of artificial dissipation terms added to the governing equation. It should be noted that all options in both codes were Verified with a single manufactured solution. The details for these tests can be found in Appendix C.

To show that MMS procedure is applicable to a more complex set of equations than Burgers equation, the Verification procedure was applied to codes that solve the time-dependent laminar 2-D incompressible and time-dependent laminar compressible Navier-Stokes equations. For the incompressible case (Section 6.1), continuity and momentum equations are solved using a finite volume approach on a staggered Cartesian mesh. For the compressible case (Section 6.2), energy equation is added to the governing equations and the coupled system is solved using a finite difference scheme on a Cartesian mesh. *Although the compressible equations are more involved numerically, the effort needed to verify the compressible code by the MMS procedure is not significantly more than the effort needed to verify the incompressible code.*

6.1 Incompressible Navier-Stokes

In this Section MMS procedure is applied to verify a code that solves the time-dependent 2-D laminar incompressible Navier-Stokes equations. The first step in the procedure is to identify what equations are solved by the code. This information can be obtained from the User's Manual or other documents that discuss the theory or the discretization method of the code. The code uses the artificial compressibility method, which is also known as pseudo-compressibility, to solve the governing equations. This approach introduces an artificial time-like derivative of pressure into the continuity equation. For the steady-state problems this time-dependent term vanishes and we recover the original continuity equation. The governing equations are given by

$$\left(\frac{1}{\beta}\right)\frac{\partial p}{\partial t} + \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = S_p \tag{26}$$

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(u^2 + \frac{p}{\rho}\right) + \frac{\partial}{\partial y}(uv) = \upsilon\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + S_u \tag{27}$$

$$\frac{\partial v}{\partial t} + \frac{\partial}{\partial x}(uv) + \frac{\partial}{\partial y}\left(v^2 + \frac{P}{\rho}\right) = \upsilon\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + S_v \tag{28}$$

where u and v are velocity components, P is the pressure, ρ is the density, ν is the kinematic viscosity, β is a constant, and S_p, S_u, S_v are the source terms generated for the manufactured solution for continuity and momentum equations, respectively. The code has no options, so only one coverage test is needed to verify the code.

Next, using the procedure described in Section 3.1 we generate a steady manufactured solution for velocity components and pressure:

$$u(x, y) = u_0[\sin(x^2 + y^2) + \varepsilon] \tag{29}$$

$$v(x, y) = v_0[\cos(x^2 + y^2) + \varepsilon] \tag{30}$$

$$P(x, y) = P_0[\sin(x^2 + y^2) + 2] \tag{31}$$

where u_0, v_0, P_0 and ε are constants. To generate the source terms, Mathematica is used to substitute the manufactured solution into the governing equations. The source terms are

$$S_p(x, y) = 2u_0 x\cos(x^2 + y^2) - 2v_0 y\sin(x^2 + y^2) \tag{32}$$

$$S_u(x, y) = \frac{1}{\rho}\{2[(P_0 x + \rho u_0(2\varepsilon u_0 x - 2\nu + \varepsilon v_0))\cos(x^2 + y^2) \tag{33}$$
$$\rho u_0(v_0 y\cos[2(x^2 + y^2)] + (2x^2\nu - \varepsilon v_0 y + 2\nu y^2 + 2u_0 x\cos(x^2 + y^2))\sin(x^2 + y^2))]\}$$

$$S_v(x,y) = \frac{1}{\rho}\{2[(\varepsilon\rho u_0 v_0 x + 2\rho v_0 x^2 \nu + P_0 y + 2\rho v_0 \nu y^2)\cos(x^2+y^2) \tag{34}$$
$$-\rho v_0(-u_0 x\cos[2(x^2+y^2)] + (\varepsilon u_0 x - 2\nu + 2\varepsilon v_0 y + 2v_0 y\cos(x^2+y^2))\sin(x^2+y^2))]\}$$

The source term for the continuity equation (Equation 32) does not contain the contribution from the pseudo-compressibility term because it vanishes as the steady-state solution is calculated. Alternatively, the pseudo-compressibility term could have been accounted for in the source term with no change in the results. Next, the computational domain is constructed using Cartesian coordinates. The Dirichlet boundary condition is used on all boundaries and its values are computed from the manufactured solution.

The finite volume scheme uses a staggered mesh (depicted in Figure 3) where the velocities are located at cell faces and the pressure at nodes. Boundary conditions are applied using ghost cells. The code outputs the solution at the nodes; bilinear interpolation is used to calculate nodal velocity components. The pressure discretization is first-order accurate. The discretization for the velocity components and the bilinear interpolation are second-order accurate.

The constants in the manufactured solution are defined as: $u_0 = 1.0$, $v_0 = 1.0$, $P_0 = 1.0$, $\rho = 1.0$, $\nu = 0.5$, $\varepsilon = 0.001$. The calculations were performed with $\beta = 40.0$ and convergence tolerance of 1.0E-12. The Cartesian computational domain, $-0.1 \le x \le 0.7$, $0.2 \le y \le 0.8$, is selected to avoid symmetry in the solution. The interior solution was initialized to 1% of the exact solution.

Five different grids, 11x9, 21x17, 41x33, 81x65, and 161x129, with grid refinement ratio of two, were used in the grid convergence test. Tables 2-4 show the computed errors based on nodal values and the observed order-of-accuracy for all the computed variables. Figures 4 and 5 show the solution of the velocity components and pressure and their corresponding error distributions. The tabulated results show second-order behavior for *u* and *v* components of velocity and first-order for the pressure. Note that these results not only verify the order-of-accuracy of the discretization scheme but also the bilinear interpolation used in interpolating the velocity components to the nodes.

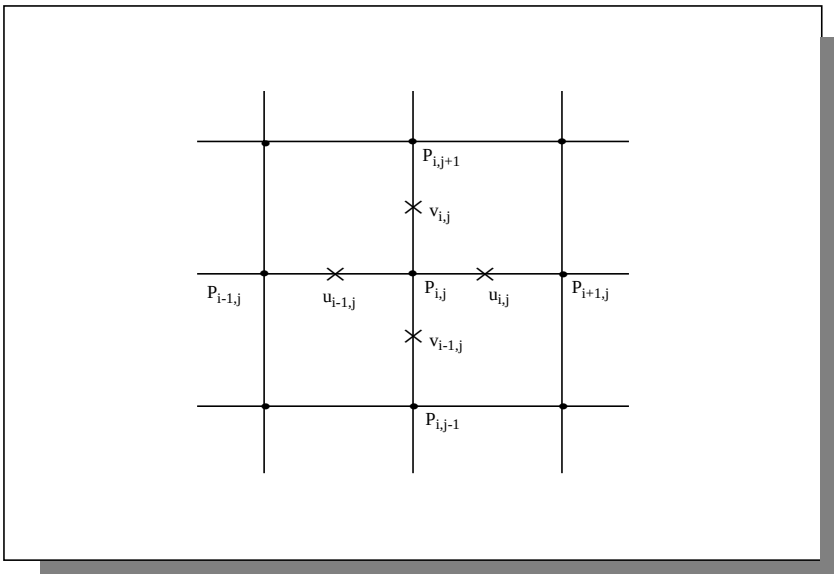

Figure 3. Staggered mesh used in the discretization of the incompressible Navier-Stokes equations

Table 2: Incompressible Navier-Stokes, Dirichlet boundaries, u-component

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	8.99809E-3			3.86235E-2		
21x17	2.04212E-3	4.41	2.14	9.65612E-3	4.00	2.00
41x33	4.79670E-4	4.26	2.09	2.43770E-3	3.96	1.99
81x65	1.15201E-4	4.16	2.06	6.09427E-4	4.00	2.00
161x129	2.80641E-5	4.10	2.04	1.52357E-4	4.00	2.00

Table 3: Incompressible Navier-Stokes, Dirichlet boundaries, v-component

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.65591E-3			4.76481E-3		
21x17	4.05141E-4	4.09	2.03	1.19212E-3	4.00	2.00
41x33	1.01449E-4	3.99	2.00	2.98088E-4	4.00	2.00
81x65	2.55309E-5	3.97	1.99	7.45257E-5	4.00	2.00
161x129	6.41893E-6	3.98	1.99	1.86316E-5	4.00	2.00

Table 4: Incompressible Navier-Stokes, Dirichlet boundaries, Pressure

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	7.49778E-4			1.36941E-3		
21x17	3.98429E-4	1.88	0.91	5.90611E-4	2.32	1.21
41x33	2.02111E-4	1.97	0.98	2.49439E-4	2.37	1.24
81x65	1.00854E-4	2.00	1.00	1.13466E-4	2.20	1.14
161x129	5.00212E-5	2.02	1.01	5.71929E-5	1.98	0.99

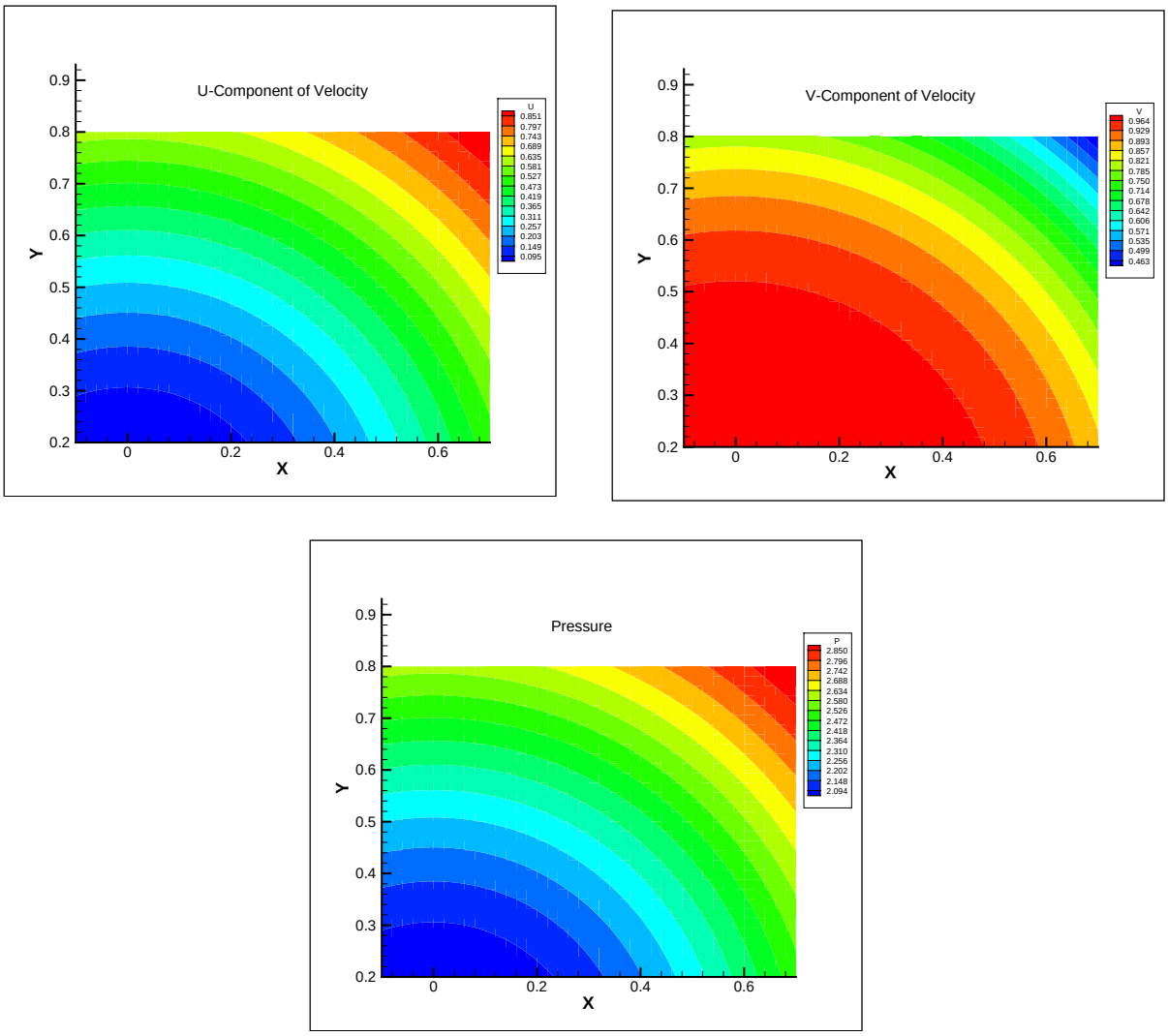

Figure 4. Manufactured solutions for the incompressible Navier-Stokes equations. Velocity components and pressure are shown.

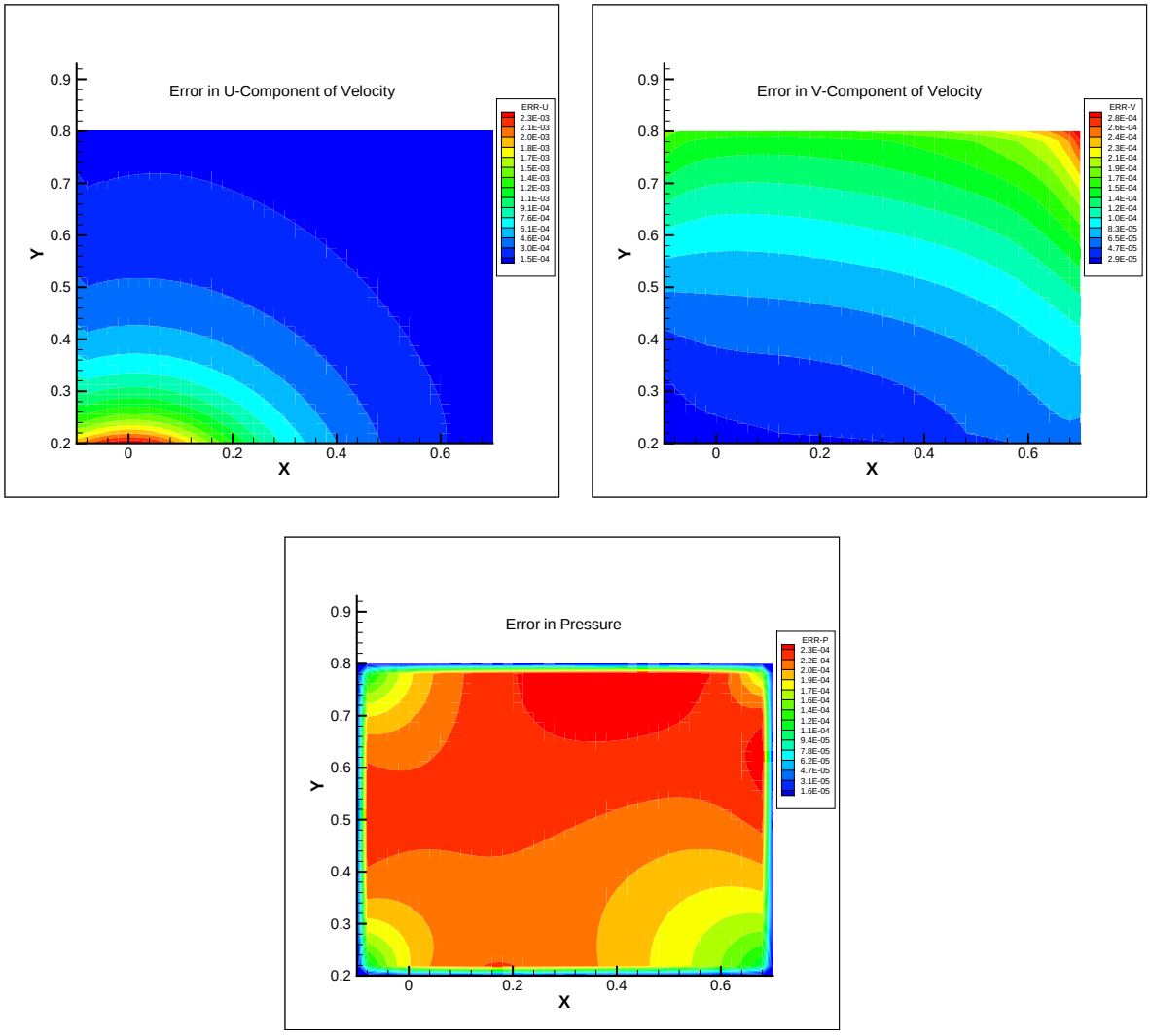

Figure 5. Solutions for the incompressible Navier-Stokes equations. Error distributions for velocity components and pressure are shown.

6.2 Compressible Navier-Stokes Equations

The purpose of this Section is to demonstrate that the MMS procedure is applicable to a more complex set of governing equations than we have seen so far. The compressible Navier-Stokes equations add a layer of complexity beyond the incompressible equations in which the energy equation is coupled to the flow equations. The code selected for Verification solves the time-dependent 2-D laminar compressible Navier-Stokes equations using finite differences on Cartesian coordinates. The code is second-order accurate in space and first-order accurate in time. The computed variables are collocated at the nodes.

The first step in the Verification process is to identify the equations that are solved by the code. In this particular code, fourth-order dissipation or smoothing terms are added to the governing equations to stabilize the numerical method. The coefficient of these terms are grid size dependent and thus, their contributions to the governing equations vanishes as $\Delta \rightarrow 0$ (see Section C.1.3 in Appendix C for more detail on added terms to the governing equations). The equations used by the code are given by:

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho u) + \frac{\partial}{\partial y}(\rho v) = C_\rho \left[(\Delta x)^4 \frac{\partial^4}{\partial x^4}(\rho) + (\Delta y)^4 \frac{\partial^4}{\partial y^4}(\rho) \right] + S_\rho \tag{35}$$

$$\begin{aligned} \frac{\partial}{\partial t}(\rho u) + \frac{\partial}{\partial x}(\rho u^2 + p) + \frac{\partial}{\partial y}(uv) &= \frac{\partial}{\partial x}(\tau_{xx}) + \frac{\partial}{\partial y}(\tau_{xy}) + S_u \\ &+ C_{\rho u} \left[(\Delta x)^4 \frac{\partial^4}{\partial x^4}(\rho u) + (\Delta y)^4 \frac{\partial^4}{\partial y^4}(\rho u) \right] \end{aligned} \tag{36}$$

$$\begin{aligned} \frac{\partial}{\partial t}(\rho v) + \frac{\partial}{\partial x}(\rho uv) + \frac{\partial}{\partial y}(\rho v^2 + p) &= \frac{\partial}{\partial x}(\tau_{xy}) + \frac{\partial}{\partial y}(\tau_{yy}) + S_v \\ &+ C_{\rho v} \left[(\Delta x)^4 \frac{\partial^4}{\partial x^4}(\rho v) + (\Delta y)^4 \frac{\partial^4}{\partial y^4}(\rho v) \right] \end{aligned} \tag{37}$$

$$\begin{aligned} &\frac{\partial}{\partial t}(\rho e_t) + \frac{\partial}{\partial x}[u(\rho e_t + p)] + \frac{\partial}{\partial y}[v(\rho e_t + p)] = \frac{\partial}{\partial x}(u\tau_{xx} + v\tau_{xy} - q_x) \\ &+ \frac{\partial}{\partial y}(u\tau_{xy} + v\tau_{yy} - q_y) + S_e + C_{\rho e} \left[(\Delta x)^4 \frac{\partial^4}{\partial x^4}(\rho e_t) + (\Delta y)^4 \frac{\partial^4}{\partial y^4}(\rho e_t) \right] \end{aligned} \tag{38}$$

where the component of the viscous stress tensor $\tau_{i,j}$ are given by

$$\tau_{xx} = \frac{2}{3}\mu \left(2\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} \right) \tag{39}$$

$$\tau_{yy} = \frac{2}{3}\mu \left(2\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x} \right) \tag{40}$$

$$\tau_{xy} = \mu \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \tag{41}$$

The Fourier's law for heat transfer by conduction is assumed, so that q can be expressed as

$$q_x = -\kappa \frac{\partial T}{\partial x} \tag{42}$$

$$q_y = -\kappa \frac{\partial T}{\partial y} \tag{43}$$

The system of equations is closed using the following equation of state for a perfect gas which relates pressure to the internal energy and the density.

$$p = (\gamma - 1)\rho e \tag{44}$$

where the total energy is defined as

$$e_t = e + 0.5v^2 \tag{45}$$

In the above equations, u and v are velocity components, T is the temperature, P is the pressure, ρ is the density, e is the internal energy, e_t is the total energy, μ is the molecular viscosity, κ is a thermal conductivity, γ is the ratio of specific heats, C_ρ, $C_{\rho u}$, $C_{\rho v}$, $C_{\rho e}$ are constants, and S_ρ, S_u, S_v, S_e are source terms defined by the manufactured solution for continuity, momentum and energy equations, respectively.

Since we have written this code primarily for this report, the only boundary condition that was implemented is the Dirichlet condition. Therefore, only one coverage test is needed to verify the code.

Next, the time-dependent manufactured solutions for density, velocity components and total energy are constructed.

$$\rho(x, y, t) = \rho_0[\sin(x^2 + y^2 + \omega t) + 1.5] \tag{46}$$

$$u(x, y, t) = u_0[\sin(x^2 + y^2 + \omega t) + \varepsilon] \tag{47}$$

$$v(x, y, t) = v_0[\cos(x^2 + y^2 + \omega t) + \varepsilon] \tag{48}$$

$$e_t(x, y, t) = e_{t0}[\cos(x^2 + y^2 + \omega t) + 1.5] \tag{49}$$

where ρ_0, u_0, v_0, e_{t0}, o and ε are constants. The source terms are constructed by substituting the manufactured solutions into the governing equations using Mathematica. The source terms are shown in Appendix B.

The Cartesian computational domain, $-0.1 \le x \le 0.7$, $0.2 \le y \le 0.8$, was selected to avoid symmetry in the solution.

Five different computational grids 11x9, 21x17, 41x33, 81x65, and 161x129, with grid refinement ratio of two (grid doubling) were constructed for grid convergence test. To perform the calculations the following constants are defined as: $u_0 = 1.0$, $v_0 = 0.1$, $\rho_0 = 0.5$, $e_{t0} = 0.5$, $\gamma = 1.4$, $R = 1.0$, $\kappa = 1.0$, $\mu = 0.3$, $\varepsilon = 0.5$, $C_s = 0.1$, and convergence tolerance of 1.0E-14. The interior solution was initialized to 1% of the exact solution.

Figures 6-7 show the computed solutions and the corresponding errors. Tables 5-8 show the behavior of the computed errors using nodal values, the l_2-norm and the maximum error, and the observed order-of-accuracy for all the computed variables. The results show that all the variables are converging second-order accurate, which matches the theoretical order-of-accuracy. Therefore, the code is Verified.

Table 5: Compressible Navier-Stokes, Dirichlet boundaries, Density

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	6.70210E-3			2.31892E-2		
21x17	8.68795E-4	7.71	2.95	3.90836E-3	5.93	2.57
41x33	1.60483E-4	5.41	2.44	7.70294E-4	5.07	2.34
81x65	3.43380E-5	4.67	2.22	2.02016E-4	3.81	1.93
161x129	7.99000E-6	4.30	2.10	5.24471E-5	3.85	1.95

Table 6: Compressible Navier-Stokes, Dirichlet boundaries, u-component

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	5.46678E-4			1.05851E-3		
21x17	1.21633E-4	4.49	2.17	2.47530E-4	4.28	2.10
41x33	2.93800E-5	4.14	2.05	6.05250E-4	4.09	2.03
81x65	7.24966E-6	4.05	2.02	1.49461E-5	4.05	2.02
161x129	1.80200E-6	4.02	2.01	3.72287E-6	4.01	2.01

Table 7: Compressible Navier-Stokes, Dirichlet boundaries, v-component

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.88554E-3			6.61668E-3		
21x17	3.09664E-4	6.09	2.61	8.57300E-4	7.72	2.95
41x33	6.14640E-5	5.04	2.33	1.35348E-4	6.33	2.66
81x65	1.33259E-5	4.61	2.21	3.16506E-5	4.28	2.10
161x129	3.12715E-6	4.26	2.09	7.61552E-6	4.16	2.06

Table 8: Compressible Navier-Stokes, Dirichlet boundaries, energy

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.15937E-4			5.12965E-4		
21x17	5.27574E-5	4.09	2.03	1.24217E-4	4.13	2.05
41x33	1.32568E-5	3.98	1.99	3.17437E-5	3.91	1.97
81x65	3.32396E-6	3.99	2.00	7.99370E-6	3.97	1.99
161x129	8.31847E-7	4.00	2.00	2.00623E-6	3.98	1.99

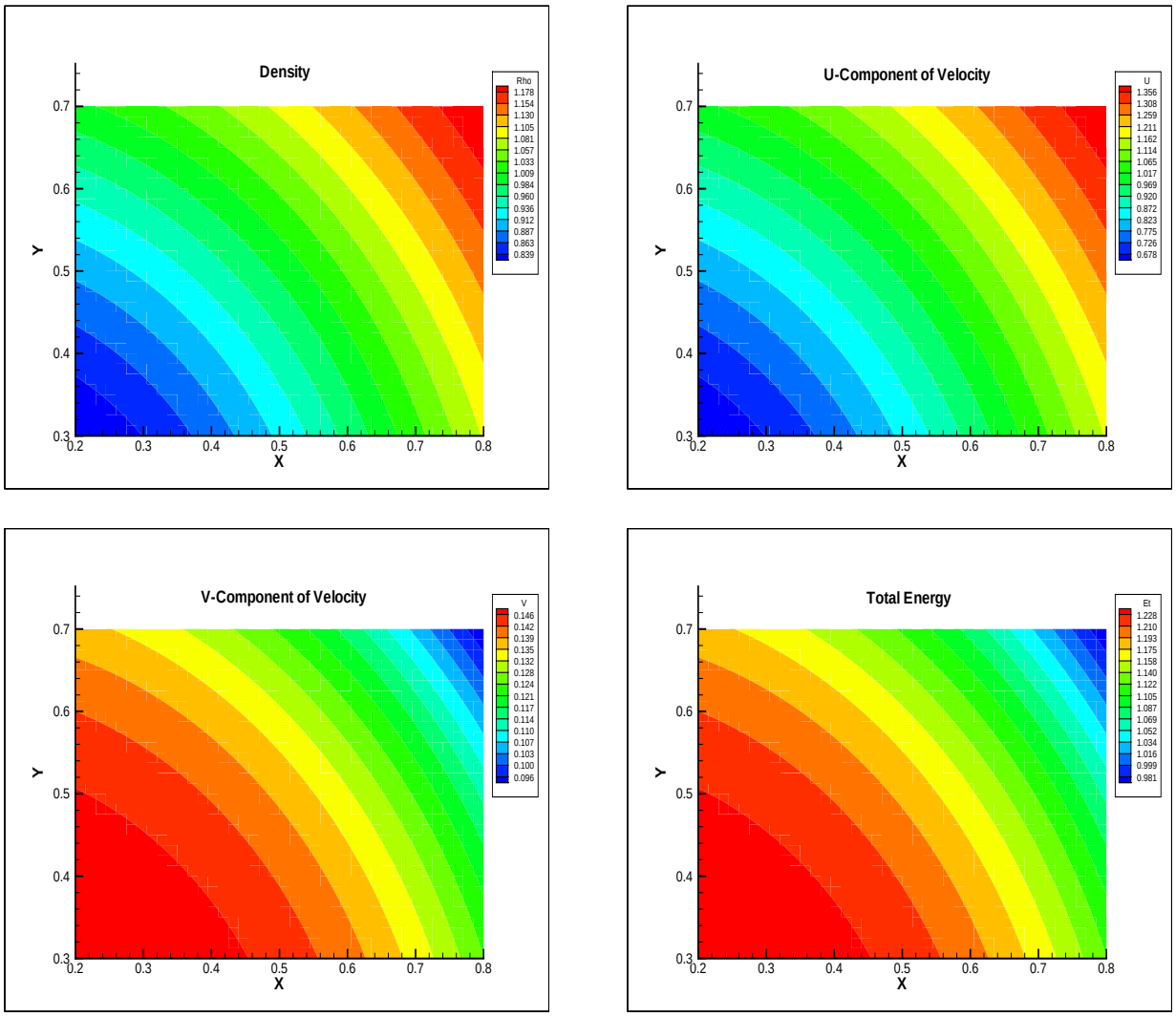

Figure 6. Manufactured solutions for the 2-D compressible Navier-Stokes equations. The variables density, u-, v-component, and total energy are shown.

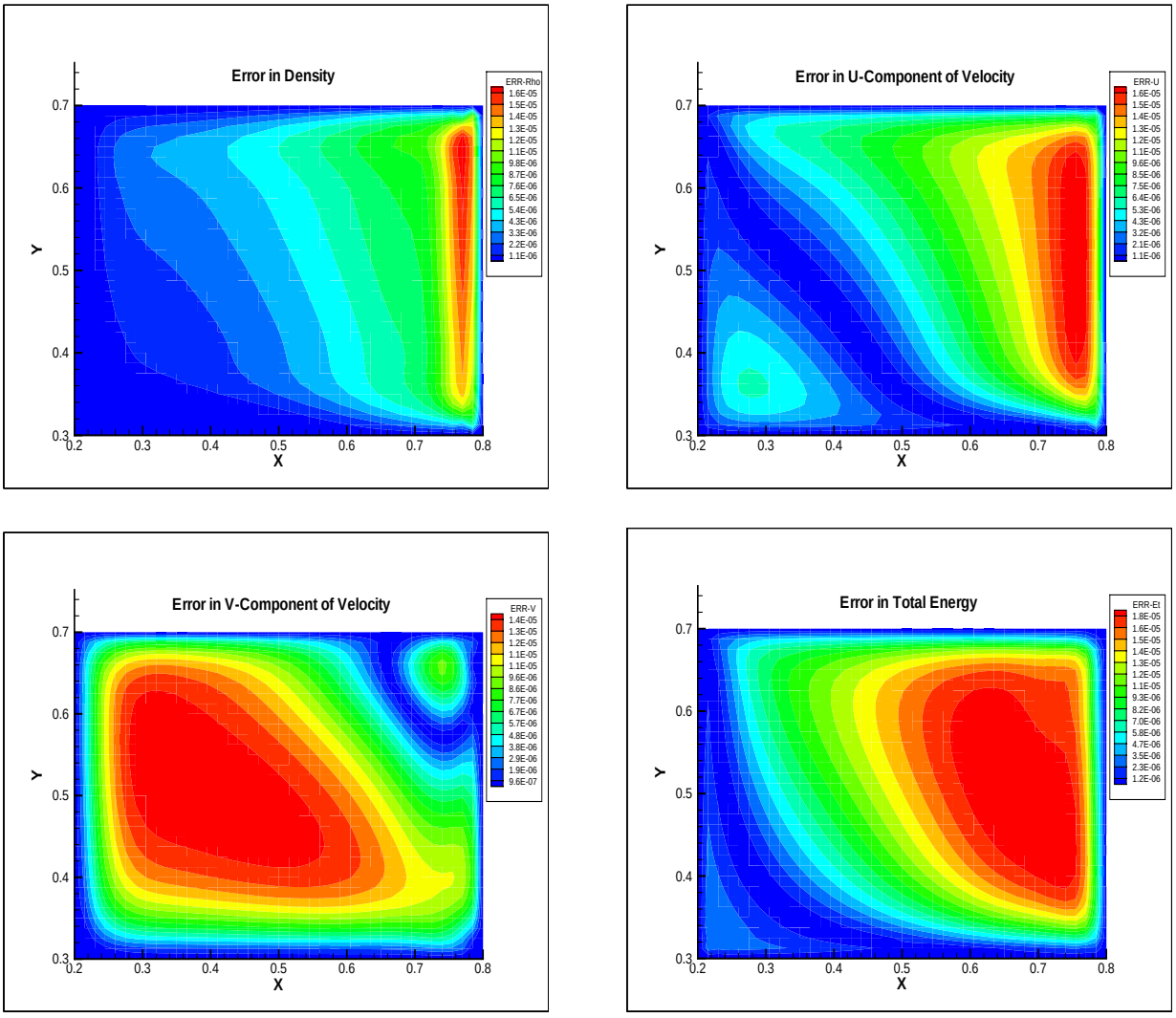

Figure 7. Solution for the 2-D compressible Navier-Stokes equations. The error in the variables, density, u-, v-component, total energy are shown.

7 Examples of Coding Mistakes that Can and Cannot be Detected by the MMS Procedure: Results of Twenty-one Blind Tests

To illustrate the kinds of coding mistakes that can and cannot be detected by the MMS procedure, we conducted a series of blind tests. The 2-D compressible Navier-Stokes code that was Verified to be second-order accurate (p=2) in Section 6.2 was given to one of the authors who intentionally altered the source code slightly to create a series of realistic coding mistakes. The other author then tried to detect the unknown mistakes using the MMS procedure. The suite of twenty-one tests was fairly comprehensive in that examples of most kinds of realistic mistakes for this code were included. In Appendix E, we describe each coding mistake and show the results of the grid convergence test. The coding mistakes created by altering the code are classified according to the taxonomy of mistakes described in Section 2.4.

Every test case except case E.13 contained a coding mistake (i.e., a change in the Verified code). Of the twenty test problems which contained mistakes, there was one Static, one Divergence, one Efficiency, ten OAM, and seven Formal mistakes.[1] *Significantly, the MMS procedure detected ten out of ten OAM's.* MMS also detected the static and divergence mistakes. The efficiency mistake and the seven formal mistakes were not detected by MMS. Thus, in this example, MMS has performed in the desired manner: *any coding mistake which prevented the governing equations from being solved correctly was detected.* None of the coding mistakes which escaped detection prevented the equations from being solved correctly.

The tests demonstrate that OAM's can arise from minor, hard-to-detect typographical mistakes which can be detected via the MMS procedure.

We observe that if the order-of-accuracy acceptance criterion were replaced with the consistency criterion, six out of ten OAM's would have been detected. These tests (E.1, E.2, E.3, E.9, E.11, and E.16) contained consistency mistakes. On the other hand, there were four OAM's that would not be detected by the consistency criterion: E.4, E.8, E.15, and E.17. To be precise, test case E.8 is an example of where inconclusive results would be obtained from the consistency criterion (more grid resolution is needed). Thus, in general, the consistency test is a less sensitive criterion but can be used if, for example, the theoretical order-of-accuracy of the code being Verified is not known. On the other hand, since both the consistency and order-of-accuracy criteria require the same data, and little extra work is required to assess the order-of-accuracy, we recommend that one use the order-of- accuracy criterion with MMS whenever possible.

1. Known coding mistakes can often be classified according to our taxonomy *a priori*.

8 Strengths and Limitations of the MMS Procedure

As with all methods, the MMS code Verification procedure has strengths and limitations. These are summarized in this Section.

8.1 Strengths

- ➢ *The majority of code capabilities can be Verified with MMS procedure* because one can construct solutions to the fully general equations solved by the code.
- ➢ For the method of manufactured solutions, governing equations which are non-linear and/ or coupled are not significantly more difficult to solve than uncoupled linear systems of equations.
- ➢ In general, the solution domain and the boundary conditions can be selected after construction of the manufactured solution to the interior equations.
- ➢ The source term can be computed using symbolic manipulations codes
- ➢ The manufactured solution is composed of easily evaluated functions.
- ➢ Solution flexibility permits one to verify code options by process of elimination.[1]
- ➢ The MMS procedure is largely insensitive to the choice of numerical method. In principle, it will work with finite differences, finite volume methods, finite element, and boundary element methods. As long as some ordered discrete approximation to the PDEs being solved is made, the method can, in principle, be applied. Likewise, the method does not depend on the type of mesh (Cartesian, polar, structured, unstructured) employed.
- ➢ Adequate care must be taken to correctly apply each step within the MMS procedure to avoid back-tracking. For example, if a careless code input mistakes were made in Step 4, it may not be caught until Step 7. Nearly all implementation mistakes and even mistakes introduced when making code modifications (e.g., to include a distributed source term) will be caught further on in the MMS procedure, i.e., the MMS procedure is largely *self-correcting*. The only known implementation mistake that would not be caught by the procedure occurs in Step 2 (Coverage Test Design). If one fails to account for all relevant code capabilities in the design of the test suite, then certain options of the code will not be Verified.

8.2 Limitations

Application of the MMS procedure is made more difficult if the governing equations lack source terms. This situation may also arise if the governing equations *do* contain source terms but the code to be Verified does not have the capability to input distributed source term data. There are two possible remedies. First, it may be possible to modify the code to incorporate a distributed source term. Often this is not a difficult task since numerical treatment of source terms is usually straightforward. This approach does raise the possibility that a coding mistake is introduced. Such a mistake would, however, be detected during the Verification procedure. In cases where one cannot add their own source term to the governing equations (e.g., proprietary software), one may take the approach suggested in Appendix B (Governing Equations with no Source Terms). In this approach, an exact solution to the *interior* equations is found. Although one has an exact solution, the auxiliary conditions are determined afterwards using the exact solution. Then the remainder of the MMS code Verification procedure can be followed.

1. In MES, code options are intimately tied to the exact solution.

An equally serious situation that could arise is illustrated by a code which contains a “look-up” table as its only constitutive relationship. In this case one must modify the code to include another constitutive relation.

Some codes may lack sufficiently general input capability to permit Verification of certain code options by the MMS procedure. In this situation, if MMS is to be applied, one is forced either to modify the code to achieve the desired generality or to forego Verification of the option. As an example, if an adaptive time-step option is to be Verified, one must modify the code to calculate the values of the boundary conditions at the time levels selected by the code.

9 Summary and Conclusions

We have described a procedure for code Verification to determine whether or not the code solves its governing equations correctly. The procedure relies on the manufacture of a fully general solution to the governing equation. Generality of the solution is essential because it guarantees that few, if any, code capabilities will be unverified. Guidelines for the construction of the manufactured solution are given in Section 3.1 and 3.2. MMS (or any other code Verification procedure) does not address the issues of code robustness, performance, or formal correctness, nor does it pertain to Solution Accuracy Assessment. Of the four acceptance criteria that were described, we advocate that MMS be used with the order-of-accuracy criterion in which one confirms the theoretical order-of-accuracy for the discretization method. Of course, when the theoretical order-of-accuracy is unknown, one can effectively apply the consistency criterion. A complete description of the MMS code Verification procedure was given in Section 5. Section 6 demonstrated the applicability of MMS to complex governing systems of equations such as the Navier-Stokes equations. Section 7 illustrated the types of coding mistakes in our taxonomy that can and cannot be detected by MMS procedure. The generality of the manufactured solution ensures that, by definition, the MMS procedure will detect any coding mistake that prevents the equations from being solved correctly. The most important known limitation of MMS procedure is the requirement that the code contain an option for input of distributed source terms and spatially-variable boundary data. If this is not the case, one may either modify the source code or, if that is not possible, rely on MES.

We have not tried MMS procedure on every possible code or situation and acknowledge that there may be additional limitations we do not know about. For example, the MMS procedure has not been rigorously applied to finite element codes, to highly specialized boundary conditions, to codes that contain grid-size dependent physical models, or to codes based on gridless methods. However, MMS procedure has been successfully applied to complex codes such as a code that solves convective transport with a moving coordinate frame [18] and to a code that solves flow with a free-surface boundary condition [19]. In Section 7 of this report we showed that it can also be successfully applied to a code solving the two dimensional Burger's equation, and to both compressible and incompressible Navier-Stokes codes. We feel confident that, with minor modifications, the MMS procedure can be applied to other situations and codes. We hope that others will apply MMS procedure to other codes and situations to help delineate its full range of applicability.

Because the MMS procedure permits, in general, Verification of more code capabilities than other Verification methods we strongly recommend that, in addition to the existing suite of Static and Dynamic code tests, the MMS procedure be adopted for ASCI code Verification[1].

1. MMS is clearly applicable to other Sandia codes as well.

References

[1] P. J. Roache, *Verification and Validation in Computational Science and Engineering*, Hermosa Publishers, Albuquerque NM, 1998.

[2] "Guide for the Verification and Validation of Computational Fluid Dynamics Simulations," AIAA G-077-1998.

[3] W.L. Oberkampf, F.G. Blottner, and D.P. Aeschliman, "Methodology for Computational Fluid Dynamics: Code Verification/Validation, AIAA 95-2226, 26th AIAA Fluid Dynamics Conference, June 19-22, 1995, San Diego.

[4] W.L. Oberkampf, F.G. Blottner, "Issues in Computational Fluid Dynamics: Code Verification and Validation," *AIAA Journal*, Vol. 36, No. 5, pp. 687-695, 1998.

[5] T.M. Shih, "A Procedure to Debug Computer Programs," *International Journal of Numerical Methods of Engineering*, Vol. 21, pp.1027-1037, 1985.

[6] S. Steinberg and P.J. Roache, "Symbolic Manipulation and Computational Fluid Dynamics", *Journal of Computational Physics*, Vol. 57, No. 2, January 1985, pp. 251-284.

[7] P.J. Roache, P.Knupp, S. Steinberg, and R.L. Blaine, "Experience with Benchmark Test Cases for Groundwater Flow," ASME FED, Vol. 93, Benchmark Test Cases for Computational Fluid Dynamics, I. Celik and C.J. Freitas, Eds., Book No. H00598-1990, pp. 49-56.

[8] P.J. Roache, "Verification of Codes and calculations," *AIAA Journal*, Vol. 36, pp. 696-702, 1998.

[9] M. Ozisik, *Heat Conduction*, John Wiley & Sons, NY, 1980.

[10] A. Jarvis and V. Crandall, *Inroads to Software Quality "How To" Guide and Toolkit*, Upper Saddle River, NJ, Prentice Hall, 1997.

[11] G. G. Schulmeyer, et al., *The Handbook of Software Quality Assurance*, Upper Saddle River, NJ, Prentice Hall, 1998.

[12] D. W. Karolak, *Software Engineering Risk Management*, IEEE Computer Society Press, Los Alamitos CA, 1996.

[13] P.L. Knepell, D.C. Arangno, *Simulation Validation: A Confidence Assessment Methodology*, IEEE Computer Society Press, Los Alamitos CA, 1993.

[14] R.O. Lewis, *Independent Verification and Validation: A Life Cycle Engineering Process for Quality Software*, Wiley, 1992.

[15] G. Emanuel, *Analytical Fluid Dynamics*, CRC Press, 1994.

[16] E. A. Sudicky and E. O. Frind, "Contaminant Transport in Fractured Porous Media: Analytical Solutions for a System of Parallel Fractures," Water Resources Research, Vol. 18, No. 6, PP. 1634-1642.

[17] Paul van Gulick, “Evaluation of Analytical Solution for a System of Parallel Fractures for Contaminant Transport in Fractured Porous Media,” Technical Memo, GeoCenters, Inc., July 1994.

[18] K.Salari, “Verification of the SECOTP-TRANSPORT Code,” SAND Report, SAND92-070014-UC-721, Preliminary Performance Assessment for the Waste Isolation Pilot Plant, December 1992.

[19] P. Knupp, “A General Solution to the free-surface equations of MWT3D via the method of manufactured solutions,” WIPP Quality Assurance Documents, 1999.

Appendix A: Example of an Exact Solution for MES

As an example of an exact solution for MES, let us apply classical methods to the heat-conduction equation (A1). For the exact solution, we solve a homogeneous heat conduction problem in a solid sphere [9]. Rewriting Equation A1 in spherical coordinates (r,θ,ϕ) and making the simplifying assumption that thermal conductivity k, density ρ, and specific head at constant pressure C_p are constants, we get:

$$\frac{\partial^2 T}{\partial r^2} + \frac{2}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2 \sin\theta}\frac{\partial}{\partial \theta}\left(\sin\theta \frac{\partial T}{\partial \theta}\right) + \frac{1}{r^2 \sin^2\theta}\frac{\partial^2 T}{\partial \phi^2} = \frac{1}{\alpha}\frac{\partial T}{\partial t} \tag{A1}$$

where $\alpha = k/(\rho C_p)$ and $T \equiv T(r, \theta, \phi, t)$. This equation is put into a more convenient form by defining the following new independent variables.

$$\begin{aligned} \mu &= \cos\theta \\ V &= r^{1/2} T \end{aligned} \tag{A2}$$

Then, equation (A1) becomes

$$\frac{\partial^2 V}{\partial r^2} + \frac{1}{r}\frac{\partial V}{\partial r} - \frac{1}{4}\frac{V}{r^2} + \frac{1}{r^2}\frac{\partial}{\partial \mu}\left[(1-\mu^2)\frac{\partial V}{\partial \mu}\right] + \frac{1}{r^2(1-\mu^2)}\frac{\partial^2 V}{\partial \phi^2} = \frac{1}{\alpha}\frac{\partial V}{\partial t} \tag{A3}$$

where we have $0 \le r < b$, $-1 \le \mu \le 1$, $0 \le \phi \le 2\pi$, for $t > 0$. Boundary conditions are:

$$\begin{aligned} V &= 0 \quad \text{at} \quad r = b, \quad \text{for} \quad t > 0 \\ V &= r^{1/2} F(r, \mu, \phi) \quad \text{for} \quad t = 0 \quad \text{in the sphere} \end{aligned} \tag{A4}$$

Using separation of variables we can construct a solution for V

$$V(r, \mu, \phi, t) = \sum_{n=0}^{\infty} \sum_{p=1}^{\infty} \sum_{m=0}^{n} e^{-\alpha \lambda_{np}^2 t} J_{n+\frac{1}{2}}(\lambda_{np} r) P_n^m(\mu) \cdot (A_{mnp} \cos m\phi + B_{mnp} \sin m\phi) \tag{A5}$$

where $J_{n+1/2}(\lambda_{np} r)$ is the solution of the Bessel's differential equation of order (n+1), $P_n^m(\mu)$ is the associated Legendre function of the first kind with n and m being positive integers. This solution satisfies the differential equation (A3) and remains bounded if the eigenvalues λ_{np} are chosen as the positive roots of

$$J_{n+\frac{1}{2}}(\lambda_{np} r) = 0 \tag{A6}$$

It also satisfies the boundary condition $r = b$. The expansion coefficients A_{mnp} and B_{mnp} are to be determined so that the initial condition for the problem is satisfied. After applying the initial condition we obtain the solution

$$T(r,\mu,\phi,t) = \frac{1}{\pi}\sum_{n=0}^{\infty}\sum_{p=1}^{\infty}\sum_{m=0}^{n}\frac{e^{-\alpha\lambda_{np}^2 t}}{N(m,n)N(\lambda_{np})}J_{n+\frac{1}{2}}(\lambda_{np}r)P_n^m(\mu)\cdot$$
$$\int_{r'=0}^{b}\int_{\mu'=\mu_0}^{1}\int_{\phi'=0}^{2\pi} r'^{3/2}J_{n+\frac{1}{2}}(\lambda_{np}r')P_n^{-m}(\mu')\cos m(\phi-\phi')F(r',\mu',\phi')d\phi' d\mu' dr' \quad \text{(A7)}$$

where the norms $N(m,n)$ and $N(\lambda_{np})$ are given by

$$N(m,n) = \left(\frac{2}{2n+1}\right)\frac{(n+m)!}{(n-m)!}$$
$$N(\lambda_{np}) = \frac{b^2}{2}\left\{J'_{n+\frac{1}{2}}(\lambda_{np}r)\right\}^2 \quad \text{(A8)}$$

In order to evaluate the above solution, we need a table of values or an algorithm to compute the Bessel function J_n and Legendre polynomial P_n, roots and their derivatives. Also, we have an infinite series, triple sum, and triple integral to compute. It should be clear that this is not an easy task. Also, this solution is not general enough (simplifying assumptions) to exercise all the variability in the coefficient of the heat-conduction equation and the computational domain is limited to a sphere.

Appendix B: Governing Equations with No Source Term

The method of manufactured solutions can be used to produce solutions to the fully general governing equation. This flexibility is mainly due to the source term. Because the source term is a code input, one is free to choose the source term as one needs, provided the source term implemented in the code allows *distributed* sources. If the code to be tested does not allow distributed sources or if the governing equations do not contain source terms, it may still be possible to construct a solution because with the code Verification *procedure* we describe, one has a great deal of latitude, having only to satisfy the interior equation. In general, however, we recommend that the code be modified to include a source term. This, of course, cannot be done if one does not have access to the source code (e.g., if one is testing commercial software). Furthermore, even if one has access to the source, one needs in addition, a good understanding of the numerical algorithm to confidently modify the code or have access to the code developers.

We give here two examples of how one can construct solutions that do not require source terms[1]. *If the governing equation is linear*, separation of variables can be effective. For example, suppose one is verifying a 2-D code which solves the heat conduction equation with no source term and a scalar conductivity:

$$\nabla \cdot k\nabla T = \alpha \frac{\partial T}{\partial t} \tag{B1}$$

with $k=k(x,y)$. Applying the separation of variables technique, let

$$T(x, y, t) = F(x, y)G(t) \tag{B2}$$

One finds that F and G satisfy

$$G' + \frac{\mu^2}{\alpha} G = 0 \tag{B3}$$

$$\nabla \cdot k\nabla F + \mu^2 F = 0 \tag{B4}$$

with $\mu \neq 0$. A solution for G is

$$G(t) = G_0 e^{-\mu^2 t/\alpha} \tag{B5}$$

To construct a solution for F, we must also construct $k(x,y)$. After a bit of trial and error, we found the following solution for μ=1:

$$\begin{aligned} F(x, y) &= e^x \cos y \\ k(x, y) &= e^x \sin y - x \end{aligned} \tag{B6}$$

1. This approach is similar to the method of exact solutions. However, with MMS we are free to determine the auxiliary conditions after the solution to the interior equations is obtained.

This solution can be scaled to ensure it satisfies the requirements outline in Sections 3.1 and 3.2. *Manufactured solutions can also be constructed for non-linear systems of homogeneous equations*. We illustrate using the equations for 2-D, steady, incompressible, laminar flow:[1]

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{B7}$$

$$\frac{\partial}{\partial x}(u^2 + h) + \frac{\partial}{\partial y}(uv) = \nu \nabla^2 u \tag{B8}$$

$$\frac{\partial}{\partial x}(uv) + \frac{\partial}{\partial y}(u^2 + h) = \nu \nabla^2 v \tag{B9}$$

with $h=P/\rho$ and ν a constant. To satisfy Equation B7, let $\phi=\phi(x,y)$ and set

$$u = -\frac{\partial \phi}{\partial y}, \qquad v = \frac{\partial \phi}{\partial x} \tag{B10}$$

Equations B8 and B9 become

$$\frac{\partial h}{\partial x} = R, \qquad \frac{\partial h}{\partial y} = Q \tag{B11}$$

where

$$\begin{aligned} R &= \frac{\partial \phi}{\partial x}\frac{\partial^2 \phi}{\partial y} - \frac{\partial \phi}{\partial y}\frac{\partial^2 \phi}{\partial x} - \nu \frac{\partial}{\partial y}(\nabla^2 \phi) \\ Q &= \frac{\partial \phi}{\partial y}\frac{\partial^2 \phi}{\partial x} - \frac{\partial \phi}{\partial x}\frac{\partial^2 \phi}{\partial y} - \nu \frac{\partial}{\partial x}(\nabla^2 \phi) \end{aligned} \tag{B12}$$

In order for h to exist, we must have

$$\frac{\partial R}{\partial y} = \frac{\partial Q}{\partial x} \tag{B13}$$

This means the ϕ must satisfy

$$\nu \nabla^4 \phi - \frac{\partial \phi}{\partial x}\frac{\partial}{\partial y}(\nabla^2 \phi) + \frac{\partial \phi}{\partial y}\frac{\partial}{\partial x}(\nabla^2 \phi) = 0 \tag{B14}$$

To construct a manufactured solution, choose ϕ so that $\nabla^2 \phi = \mu$ (a constant). Then, Equation B14 is automatically satisfied, then u and v are computed from Equation B10. Next, $h(x,y)$ can be found by computing R and Q in Equation B12, then performing an integration of Equation B11.

1. The solution constructed in this report is original and, to our knowledge, does not appear in the literature.

For example, let

$$\phi(x, y) = e^x \cos y - e^y \sin x \tag{B15}$$

Then, $\nabla^2\phi = 0$, and then, from Equation B10

$$\begin{aligned} u(x, y) &= e^x \sin y + e^y \sin x \\ v(x, y) &= e^x \cos y - e^y \cos x \end{aligned} \tag{B16}$$

From Equation B12

$$\begin{aligned} R(x, y) &= -e^{2x} - e^{x+y}[\sin(x + y) - \cos(x + y)] \\ Q(x, y) &= -e^{2y} - e^{x+y}[\sin(x + y) - \cos(x + y)] \end{aligned} \tag{B17}$$

and finally from Equation B13

$$h(x, y) = -\frac{1}{2}e^{2x} - \frac{1}{2}e^{2y} + e^{x+y}\cos(x + y) \tag{B18}$$

Appendix C: Application of the MMS procedure to a code that solves the Two-Dimensional Burgers Equation

C.1 Two-Dimensional Burgers Equation, Cartesian Coordinates

The Burgers equation provide a simple nonlinear model which is similar to the equations governing fluid flow. The code that we are going to verify solves the steady and unsteady Burgers equation in Cartesian coordinates. The code uses a centered finite difference formulation for spatial derivatives and two-point backward explicit Euler for time derivatives. The code is second-order accurate in space and first-order accurate in time. The code has options for two boundary condition types, Dirichlet and Neumann. The variables are collocated at nodes. The first step in the MMS procedure is to identify the governing equations used by the code. The two-dimensional time dependent Burgers equation in conservation form, as used by the code, is given by

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(u^2) + \frac{\partial}{\partial y}(uv) = \upsilon\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + S_u \tag{C1}$$

$$\frac{\partial v}{\partial t} + \frac{\partial}{\partial x}(uv) + \frac{\partial}{\partial y}(v^2) = \upsilon\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + S_v \tag{C2}$$

where S_u and S_v are source terms associated with the manufactured solution. Using the procedure described in Section 3 we generated manufactured solutions for the time dependent Burgers equation. The following are the manufactured solution for the u and v components of velocity:

$$u(x, y, t) = u_0[\sin(x^2 + y^2 + \omega t) + \varepsilon] \tag{C3}$$

$$v(x, y, t) = v_0[\cos(x^2 + y^2 + \omega t) + \varepsilon] \tag{C4}$$

where u_0, v_0, ω, and ε are constants. The source terms are:

$$\begin{aligned} S_u(x, y, t) = {} & u_0\{2v_0 y\cos[2(\omega t + x^2 + y^2)] + 2(2x^2\upsilon - \varepsilon v_0 y + 2\upsilon y^2)\sin(\omega t + x^2 + y^2) \\ & + [\omega + 4\varepsilon u_0 x - 4\upsilon + 2\varepsilon v_0 y + 4u_0 x\sin(\omega t + x^2 + y^2)]\cos(\omega t + x^2 + y^2)\} \end{aligned} \tag{C5}$$

$$\begin{aligned} S_v(x, y, t) = {} & -v_0\{-2u_0 x\cos[2(\omega t + x^2 + y^2)] - 2[\varepsilon u_0 x + 2\upsilon(x^2 + y^2)]\cos(\omega t + x^2 + y^2) \\ & + [\omega + 2\varepsilon u_0 x - 4\upsilon + 4\varepsilon v_0 y + 4v_0 y\cos(\omega t + x^2 + y^2)]\sin(\omega t + x^2 + y^2)\} \end{aligned} \tag{C6}$$

Mathematica was used to generate the source terms. The steady solution to Burgers equation (Equations C1 and C2) is obtained by setting the ω to zero in the manufactured solution.

The next step in the Verification process is to construct coverage tests. Here, we are interested only to verify the steady option of the code with two different boundary conditions. Two coverage tests are required as a minimum to verify the interior equations and the boundary conditions (see Section 5). The difficulty with this approach is that if the code fails the first coverage test, the coding mistake or mistakes could be in either the interior equations or the boundary conditions or both. we recommend that the first coverage test be used to only verify the interior equations. This adds one

additional coverage tests but separates the Verification of the interior equations from the boundary conditions.

In the following subsections we will discuss each coverage test in more detail.

C.1.1 Steady Solution, Dirichlet Boundaries

Since with the MMS procedure the size of the computational domain is arbitrary, we selected a region in the solution space $-0.1 \le x \le 0.7$, $0.2 \le y \le 0.8$ to avoid symmetry. *The symmetry in the solution can potentially hide coding mistakes and should be avoided.* The computational grids are dimensioned such that the maximum number of points in each coordinate direction is not the same. Again, this is done to check if there are any coding mistakes associated with the maximum number of elements or cells in the computational domain. To initialize the solution there is a temptation to use the exact solution. In most cases, this would minimize the number of iterations to convergence; however, *this should be avoided.* As will be illustrated in Section E.4, this can hide coding mistakes.

For the grid convergence study we use five different grids, 11x9, 21x17, 41x33, 81x65, and 161x129, with constant grid refinement ratio of 2 (grid doubling). The constants in the manufactured solutions are: $\nu = 0.7$, $u_0 = 1.0$, $v_0 = 1.0$, $\varepsilon = 0.001$, We have selected the convergence tolerance of 1.0E-14 for the solver. The reason for the tight tolerance is to eliminate roundoff errors due to lack of iterative convergence.

Figure C1 shows the manufactured solution on a 41x33 cartesian grid for both components of velocity and the corresponding error distributions. Tables C1 and C2 show the convergence behavior of the code for the u- and v-components of velocity based on the normalized l_2-norm of the error and the maximum error. In these tables Column 1 is the size of the grids, Column 2 is the normalized l_2-norm of the error, Column 3 is the ratio of errors in Column 2, Column 4 is the order-of-accuracy (Equation 25), Column 5 is the maximum error in the computational domain, and Columns 6-7 are the ratio of the maximum error and the order-of-accuracy. It is important to monitor the convergence behavior of both the maximum error and RMS error in order to estimate the order-of-accuracy and trace coding mistakes. It is possible that these two parameters converge toward the correct answer from the opposite side, for example one might start higher than the expected order and the other one lower; however both should converge to the same value. Tables C1 and C2 clearly show that both components of velocity in the l_2-norm and the maximum errors are converging second-order accurate. This verifies the interior equations and the Dirichlet boundary condition.

Table C1: Burgers Equation, Dirichlet boundaries, u-component

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.77094E-4			6.49740E-4		
21x17	6.56290E-5	4.22	2.08	1.66311E-4	3.91	1.97
41x33	1.59495E-5	4.11	2.04	4.17312E-5	3.99	1.99
81x65	3.93133E-6	4.06	2.02	1.04560E-5	3.99	2.00
161x129	9.75920E-7	4.03	2.01	2.61475E-6	4.00	2.00

Table C2: Burgers Equation, Dirichlet boundaries, v-component

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.16758E-4			4.00655E-4		
21x17	5.12127E-5	4.23	2.08	1.03315E-4	3.88	1.96
41x33	1.24457E-5	4.11	2.04	2.58554E-5	4.00	2.00
81x65	3.05781E-6	4.06	2.02	6.47093E-6	4.00	2.00
161x129	7.61567E-7	4.03	2.01	1.61783E-6	4.00	2.00

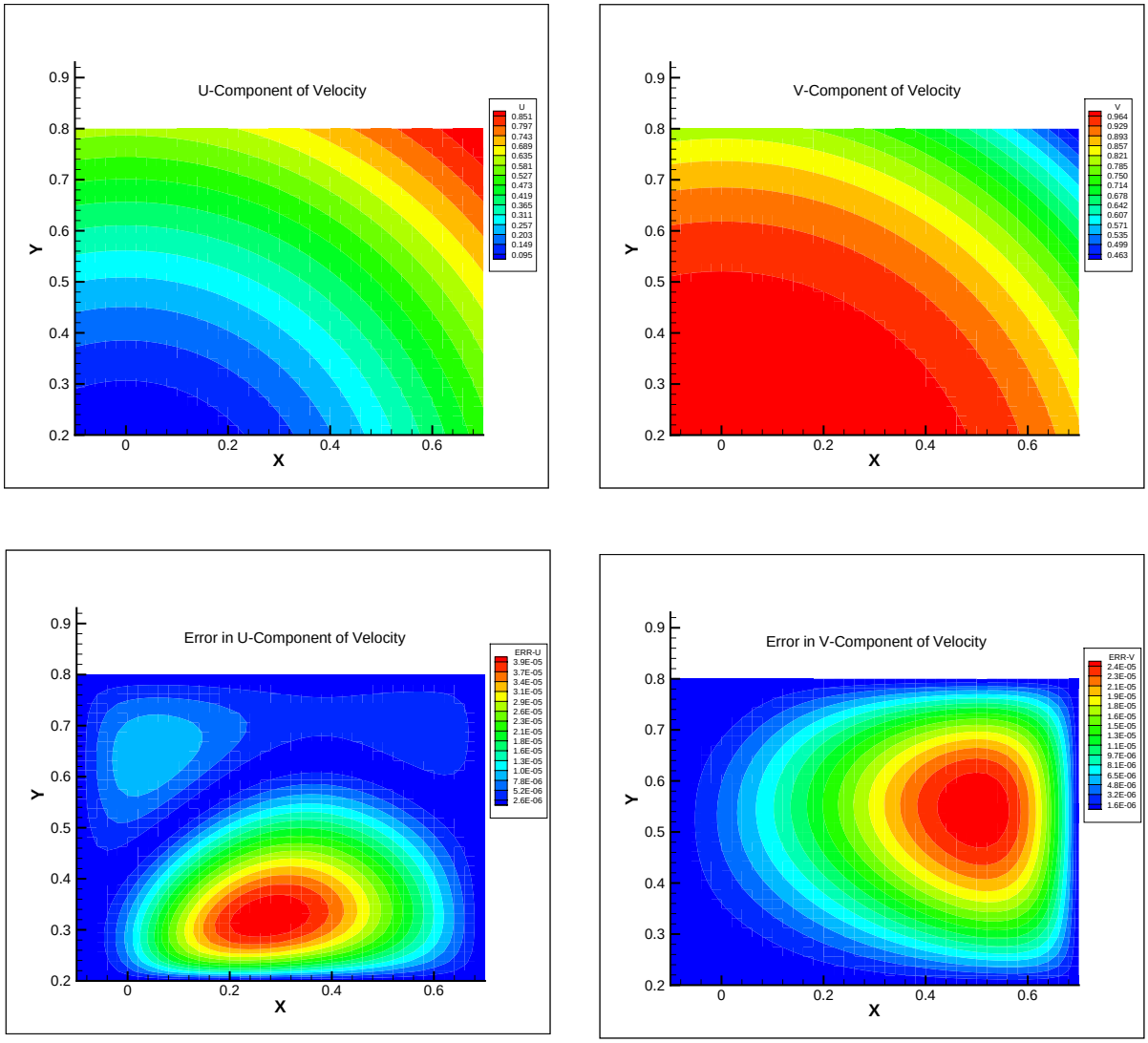

Figure C1. Two-dimensional solution of Burgers equation in Cartesian coordinates. Solutions for both components of velocity and their corresponding error distributions are shown.

C.1.2 Steady Solution, Mixed Neumann and Dirichlet Boundary Conditions

In this Section we will show how to verify the Neumann boundary condition option in the code. In the pervious Section, using the first coverage test, we Verified the interior equations with the Dirichlet boundary condition. The remaining coverage tests are used to verify the Neumann boundary condition. In the second coverage test, we assigned Neumann boundary condition at *x(1)* and *x(imax)* and the rest to Dirichlet boundary conditions as depicted in Figure C2. The Neumann conditions were given by $\partial u/\partial x = e(x, y)$, and $\partial v/\partial x = f(x, y)$, where $e(x,y)$ and $f(x,y)$ are inputs to the code. These functions are computed by evaluating the gradient of the manufactured solution. Tables C3 and C4 show the convergence behavior for the u- and v-component of velocity. It is clear from these tables that both computed variables are converging second-order accurate. In the third coverage test, the location of the Neumann and the Dirichlet boundaries were switched. In this case, the Neumann conditions were given by $\partial u/\partial y = g(x, y)$, and $\partial v/\partial y = h(x, y)$, where again, $g(x,y)$ and $h(x,y)$ are computed from the manufactured solution. Tables C5 and C6 show second-order accurate convergence behavior for both computed variables. Since the code was previously Verified with the Dirichlet boundary condition, we can interpret this result to mean that the Neumann boundary condition is working correctly and it is second-order accurate.

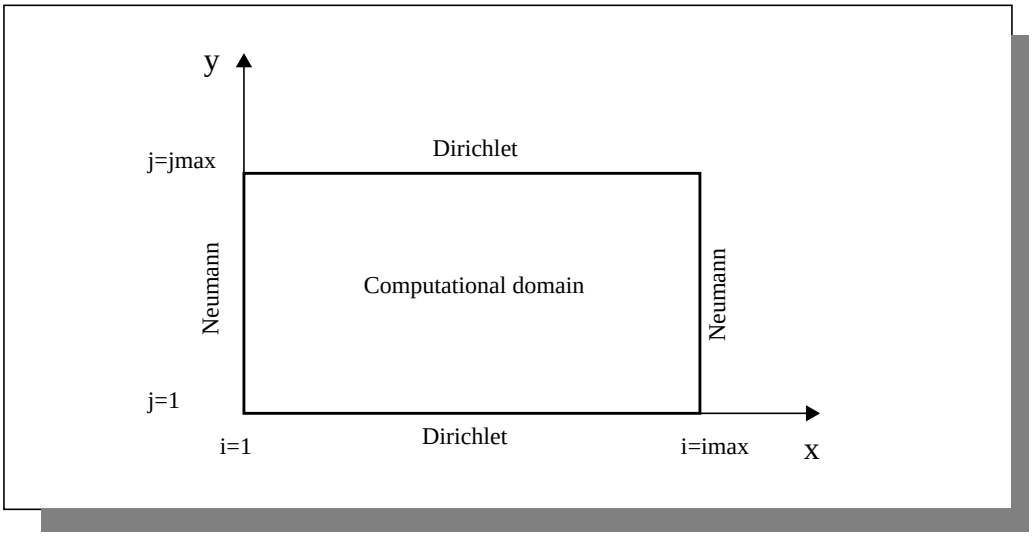

Figure C2. Schematic of the computational domain for the solution of the Burger equation.

Table C3: Burgers Equation, horizontal Neumann Boundaries, u-component

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.67064E-3			3.94553E-3		
21x17	3.81402E-4	4.38	2.13	1.04172E-3	3.79	1.92
41x33	9.19754E-5	4.15	2.05	2.70365E-4	3.85	1.95
81x65	2.26653E-5	4.06	2.02	6.89710E-5	3.92	1.97
161x129	5.63188E-6	4.02	2.01	1.74315E-5	3.96	1.98

Table C4: Burgers Equation, horizontal Neumann Boundaries, v-component

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	5.04918E-4			1.53141E-3		
21x17	8.62154E-5	5.86	2.55	3.353-7E-4	4.57	2.19
41x33	1.78470E-5	4.83	2.27	7.83351E-5	4.28	2.10
81x65	4.08392E-6	4.37	2.13	1.89318E-5	4.14	2.05
161x129	9.79139E-7	4.17	2.06	4.65444E-6	4.07	2.02

Table C5: Burgers Equation, vertical Neumann Boundaries, u-component

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.64865E-3			7.00155E-3		
21x17	9.22780E-4	3.95	1.98	2.20261E-3	3.18	1.67
41x33	2.30526E-4	4.00	2.00	6.08147E-4	3.62	1.86
81x65	5.75741E-5	4.00	2.00	1.58641E-4	3.83	1.94
161x129	1.43857E-5	4.00	2.00	4.05035E-5	3.92	1.97

Table C6: Burgers Equation, vertical Neumann Boundaries, v-component

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.49038E-3			2.69651E-3		
21x17	2.97292E-4	5.01	2.33	6.08252E-4	4.43	2.15
41x33	6.65428E-5	4.47	2.16	1.44098E-4	4.22	2.08
81x65	1.56572E-5	4.22	2.08	3.50711E-5	4.11	2.04
161x129	3.83559E-6	4.11	2.04	8.65109E-6	4.05	2.02

C.1.3 Added Terms to the Governing Equations

Some numerical methods require artificial dissipation to be added to the solution for stability purposes. In this Section we will investigate the influence of these terms on the manufactured solution. Typically, 4th-order terms are used to introduce dissipation or smoothing. We have added 4th-order terms to the Burger equations, such as:

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(u^2) + \frac{\partial}{\partial y}(uv) = \upsilon\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + S_u - C_u\left((\Delta x)^4\frac{\partial^4 u}{\partial x^4} + (\Delta y)^4\frac{\partial^4 u}{\partial y^4}\right) \tag{C7}$$

$$\frac{\partial v}{\partial t} + \frac{\partial}{\partial x}(uv) + \frac{\partial}{\partial y}(v^2) = \upsilon\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + S_v - C_v\left((\Delta x)^4\frac{\partial^4 v}{\partial x^4} + (\Delta y)^4\frac{\partial^4 v}{\partial y^4}\right) \tag{C8}$$

where C_u and C_v are constants. By design these added terms are grid size dependent and thus, their contribution to governing equations vanishes as $\Delta \rightarrow 0$. In this example, these terms are sixth-order accurate because the fourth derivatives are approximated by second-order accuracy differences. Therefore, the overall set of equations is second-order accurate. Provided the constants C_u and C_v are 'small', the effect of these terms on the solution will decrease faster than the physical terms in the equations, guaranteeing one will converge to the solution to the physical equation (provided there are no coding mistakes). To illustrate this we use the original manufactured solution Equations C3-C4 with the corresponding source terms Equations C5-C6. Since the added terms are grid size dependent there is no need to modify the original source terms.

For this exercise we set $C_u = C_v = 0.01$ and the other constants are the same as defined in the previous Sections. Tables C7 and C8 show the behavior of the computed errors and the observed order-of-accuracy for the u- and the v-component of velocity. From these tables it is clear that both components of velocity are second-order accurate. This indicates that the added terms did not alter the observed order-of-accuracy of the code as expected.

Some additional observations can be made. First, the source term generated as part of the manufactured solution does not need to account for the artificial dissipation term because it is higher order accurate than the physical terms. One the other-hand, if desired, one *could* account for

the artificial dissipation in the source term if desired, however, nothing is gained by doing so. We can generalize this observation by noting that the source term need only account for all the physically meaningful terms in the equation. In addition, it must account for non-physical terms whose order-of-accuracy is less than the accuracy of the physical terms (of course, one should never introduce such terms).

Table C7: Burgers Equation, Dirichlet boundaries, u-component, added terms

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.76688E-4			6.48866E-4		
21x17	6.56001E-5	4.22	2.08	1.66252E-4	3.90	1.96
41x33	1.59476E-5	4.11	2.04	4.17171E-5	3.99	1.99
81x65	3.93120E-6	4.06	2.02	1.04557E-5	3.99	2.00
161x129	9.75887E-7	4.03	2.01	2.61467E-6	4.00	2.00

Table C8: Burgers Equation, Dirichlet boundaries, v-component, added terms

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.16866E-4			4.00813E-4		
21x17	5.12199E-5	4.23	2.08	1.03325E-4	3.88	1.96
41x33	1.24462E-5	4.12	2.04	2.58560E-5	4.00	2.00
81x65	3.06783E-6	4.06	2.02	6.47095E-6	4.00	2.00
161x129	7.61550E-7	4.03	2.01	1.61780E-6	4.00	2.00

C.2 Two-Dimensional Burgers Equation, Curvilinear Coordinates

In this Section we demonstrate that the manufactured solution is independent of the discretization methods used to solve the governing equations. To illustrate this, we are going to verify a code that solves the time-dependent Burgers equation in the curvilinear coordinates. This is similar to a code that was Verified in Section C.1 that solved the same equations in the Cartesian coordinates. In this approach, physical domain is mapped to a computational domain through a general coordinate transformation. The code is spatially second-order accurate and temporally first-order accurate. The code is considered to be spatially second-order accurate only if the metrics of the coordinate transformation and the numerical scheme are both second-order accurate. The variables are collocated at nodes. There is only one boundary condition available which is Dirichlet boundary. The first step in the MMS procedure is to identify the governing equations. This has been done in Section C.1. Two coverage tests are needed since the code has two options for steady and unsteady solutions with Dirichlet boundary condition. The next step is to construct manufactured solutions for both component of velocity. This step is unnecessary, since the manufactured solutions that were constructed in Section C.1 still valid. In the following subsections we will present Verification results for steady and unsteady options of the code.

C.2.1 Steady Solution

For the computational domain we have used the space between two concentric circles with the center positioned at X_0 = 0.4 and Y_0 = 0.2. The inner and the outer radii are 0.15 and 0.7, respectively. The constructed grids are uniform with no stretching along the boundaries. Again, it is worth emphasizing that we should construct the computational domain to avoid symmetry in the solution. Five different grids were used for the grid refinement study (r=2). The constants in the manufactured solution are: ν = 0.7, u_0 = 1.0, v_0 = 1.0, ε = 0.001, and we set the tolerance of 1.0E-14 for the solver. Figure C3 shows the manufactured solution and the error distribution for u- and v-component of velocity. Tables C9 and C10 show the second-order behavior for all variables. Thus, we have Verified the code for the steady option with Dirichlet boundary conditions. This procedure not only Verified the solution technique but also the transformation procedure. Note that the grid for this exercise was curvilinear but orthogonal. This was not a particularly good choice since the cross-derivative terms in the metrics are zero and if we had any mistakes in those terms they will not show up in the grid convergence test. So, the better choice would have been the curvilinear and non orthogonal grid.

This exercise clearly demonstrates that a manufactured solution can be used to verify different codes with different numerical schemes. *Note that the manufactured solution would retain all its properties regardless of what code is using it.*

Table C9: Burgers Equation, Curvilinear Coordinate, Dirichlet boundaries, u-component

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.62534E-3			8.16627E-3		
21x17	8.76306E-4	4.14	2.05	2.12068E-3	3.85	1.95
41x33	2.13780E-4	4.10	2.04	5.32141E-4	3.99	1.99
81x65	5.27317E-5	4.05	2.02	1.33219E-4	3.99	2.00
161x129	1.30922E-5	4.03	2.01	3.33411E-5	4.00	2.00

Table C10: Burgers Equation, Curvilinear Coordinate, Dirichlet boundaries, v-component

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.46560E-3			2.80098E-3		
21x17	3.45209E-4	4.25	2.09	7.12464E-4	3.93	1.98
41x33	8.38134E-5	4.12	2.04	1.78464E-4	3.99	2.00
81x65	2.06544E-5	4.06	2.02	4.46808E-5	3.99	2.00
161x129	5.12701E-6	4.03	2.01	1.11731E-5	4.00	2.00

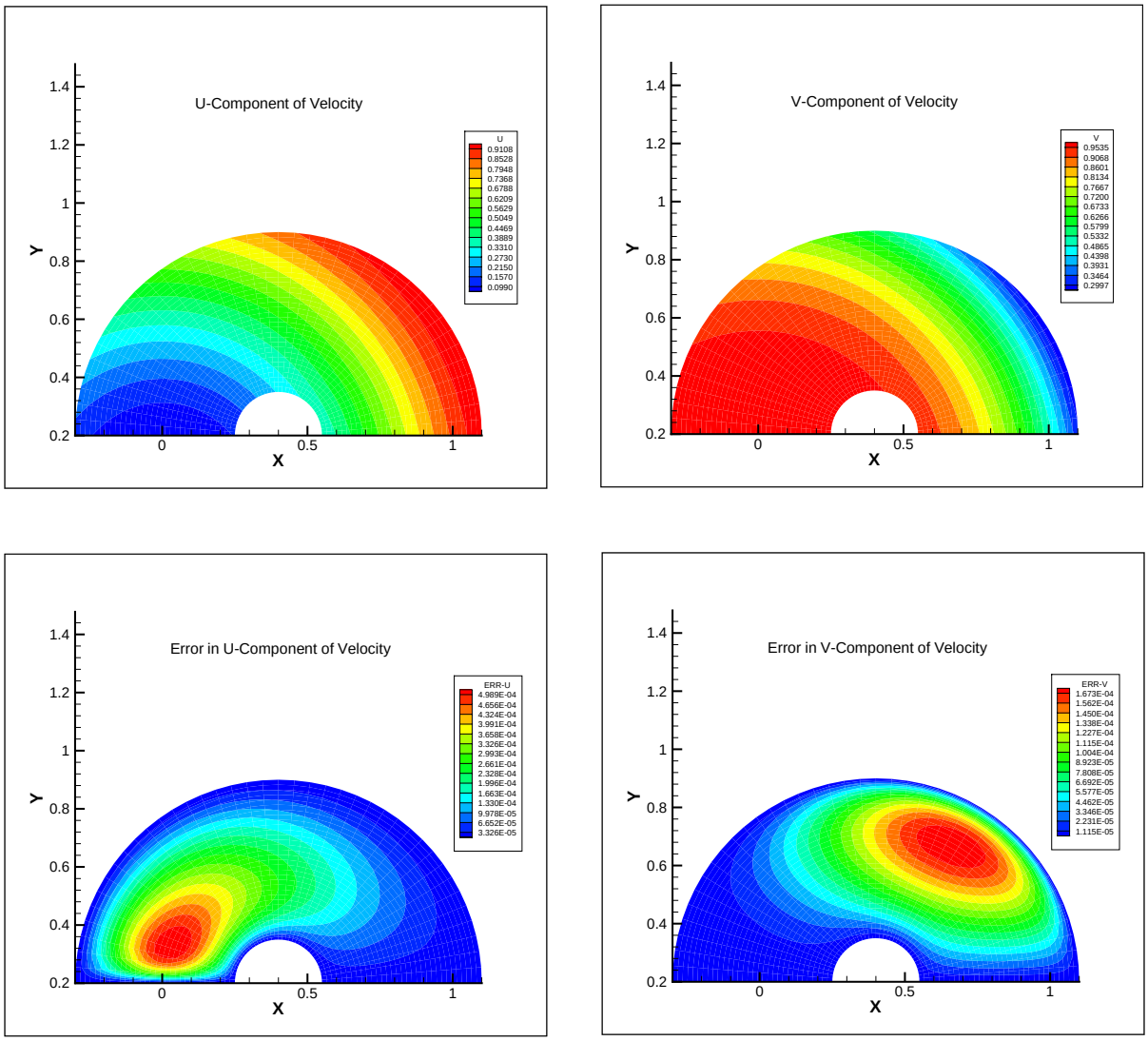

Figure C3. Two-dimensional solution of Burgers equation in curvilinear coordinates. Solutions for both components of velocity and their corresponding error distributions are shown.

C.2.2 Unsteady Solution

In the previous Sections we tested the code for the steady solution and Verified that is spatially second-order accurate. To check the code for time accuracy we have two options: one is to pick a grid that provides a solution to the governing equations for which the spatial errors are nearly reduced to machine roundoff and then refine the time step for the grid convergence test; second is to refine both the space (grid) and time together. Both approaches should provide the same order-of-accuracy for time. Since we have Verified the code for steady solution we have an idea about the grid resolution and the asymptotic range. So, we have selected to fix the grid at the size of 161x129 and refine the time step. We use five different Δt's in the convergence test. In each calculation, the flow is initialized to the same state, and we stop all calculations at a time of 8.0E-4 seconds and then compute the errors. Tables C11 and C12 show the convergence behavior of u- and v-components of velocity in time. Both variables show first-order accuracy in time which matched the expected order-of-accuracy. Given this result, we have Verified this code for both steady and unsteady solutions. Figure C4 shows the error distribution for the u- and the v-component of velocity at time of 8.0E-4 seconds. As we know, the error distribution should be different compared to the steady solution shown in Figure C3, and in fact, they are completely different. Note that, when you fix the grid and refine the time step *it is very important that the selected grid provide a solution of the governing equations for which the spatial errors are nearly reduced to machine roundoff. Otherwise, this approach will not work and one has to go back to refining the grid and the time step together.*

Table C11: Burgers Equation, Curvilinear Coordinate, Time Dependent Dirichlet boundaries, u-component

Δt	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
8.0E-6	2.70229E-3			9.62285E-3		
4.0E-6	9.44185E-4	2.86	1.52	3.15348E-3	3.05	1.61
2.0E-6	4.12356E-4	2.29	1.20	1.34886E-3	2.34	1.23
1.0E-6	1.94240E-4	2.12	1.09	6.42365E-4	2.10	1.07
0.5E-6	9.45651E-5	2.05	1.04	3.14321E-4	2.04	1.03

Table C12: Burgers Equation, Curvilinear Coordinate, Time Dependent Dirichlet boundaries, v-component

Δt	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
8.0E-6	3.82597E-4			9.72082E-4		
4.0E-6	1.46529E-4	2.61	1.38	3.98178E-4	2.44	1.29
2.0E-6	6.59684E-5	2.22	1.15	1.83016E-4	2.18	1.12
1.0E-6	3.14993E-5	2.09	1.07	8.81116E-5	2.08	1.05
0.5E-6	1.54609E-5	2.04	1.03	4.33979E-5	2.03	1.02

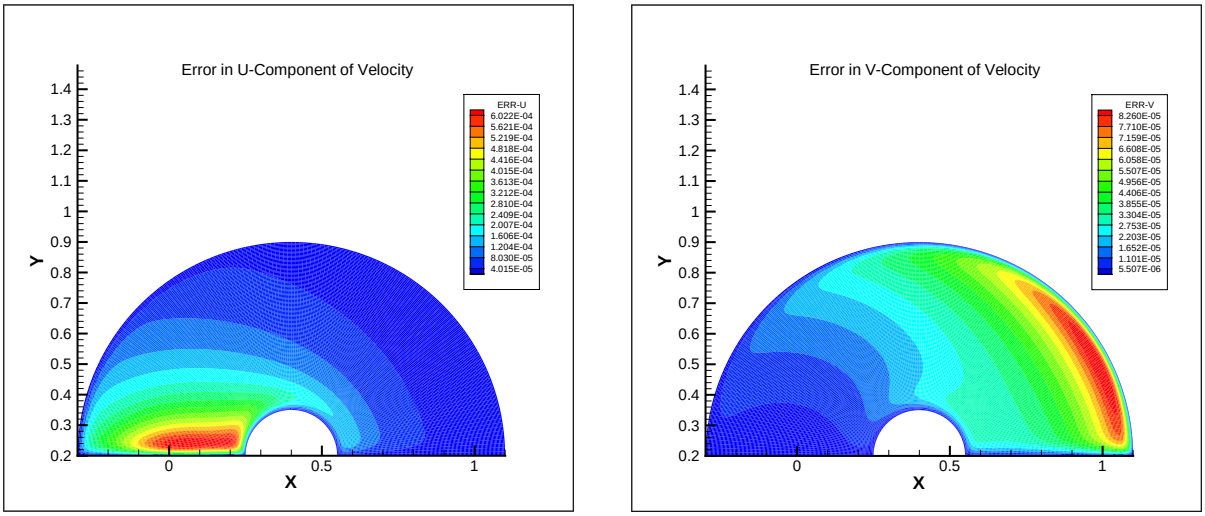

Figure C4. Unsteady two-dimensional solution of Burgers equation in curvilinear coordinates. Error distribution for u- and v-components of velocity are shown.

Appendix D: Examples of MMS Source Terms

Source terms for manufactured solutions created for the 2-D laminar compressible Navier-Stokes equations, Section 6.2.

Continuity equation

```
Out[46]= rho0 (v0 y (2 Cos[2 (omg t + x^2 + y^2)] - 3 Sin[omg t + x^2 + y^2]) +
           Cos[omg t + x^2 + y^2] (omg + 3 u0 x + 2 eps u0 x + 2 eps v0 y + 4 u0 x Sin[omg t + x^2 + y^2]))
```

Momentum equations

x-component

$$
\begin{aligned}
\text{Out[27]} = {} & \text{omg rho0 u0 Cos}[\text{omg t} + x^2 + y^2]\left(\frac{3}{2} + \text{Sin}[\text{omg t} + x^2 + y^2]\right) + \\
& \text{omg rho0 u0 Cos}[\text{omg t} + x^2 + y^2]\,(\text{eps} + \text{Sin}[\text{omg t} + x^2 + y^2]) + \\
& 2\,\text{rho0 u0 v0 y Cos}[\text{omg t} + x^2 + y^2]\,(\text{eps} + \text{Cos}[\text{omg t} + x^2 + y^2])\,(\text{eps} + \text{Sin}[\text{omg t} + x^2 + y^2]) + \\
& 2\,\text{rho0 u0}^2\,\text{x Cos}[\text{omg t} + x^2 + y^2]\,(\text{eps} + \text{Sin}[\text{omg t} + x^2 + y^2])^2 + \\
& \text{rho0 u0 v0 y Cos}[\text{omg t} + x^2 + y^2]\,(\text{eps} + \text{Cos}[\text{omg t} + x^2 + y^2])\,(3 + 2\,\text{Sin}[\text{omg t} + x^2 + y^2]) + \\
& 2\,\text{rho0 u0}^2\,\text{x Cos}[\text{omg t} + x^2 + y^2]\,(\text{eps} + \text{Sin}[\text{omg t} + x^2 + y^2])\,(3 + 2\,\text{Sin}[\text{omg t} + x^2 + y^2]) - \\
& \text{rho0 u0 v0 y Sin}[\text{omg t} + x^2 + y^2]\,(\text{eps} + \text{Sin}[\text{omg t} + x^2 + y^2])\,(3 + 2\,\text{Sin}[\text{omg t} + x^2 + y^2]) - \\
& \frac{8}{3}\,\text{xmu}\,(\text{u0} + \text{v0 x y})\,\text{Cos}[\text{omg t} + x^2 + y^2] - 2\,\text{u0}\,x^2\,\text{Sin}[\text{omg t} + x^2 + y^2]) + \\
& 2\,\text{xmu}\,(-(\text{u0} - 2\,\text{v0 x y})\,\text{Cos}[\text{omg t} + x^2 + y^2] + 2\,\text{u0}\,y^2\,\text{Sin}[\text{omg t} + x^2 + y^2]) + \\
& 2\,(-1 + \text{gam})\,\text{rho0 x}\left(\frac{3}{2} + \text{Sin}[\text{omg t} + x^2 + y^2]\right)(\,(-\text{et0} + \text{eps v0}^2)\,\text{Sin}[\text{omg t} + x^2 + y^2] - \\
& \quad \text{Cos}[\text{omg t} + x^2 + y^2]\,(\text{eps u0}^2 + (\text{u0}^2 - \text{v0}^2)\,\text{Sin}[\text{omg t} + x^2 + y^2])) + \\
& 2\,(-1 + \text{gam})\,\text{rho0 x Cos}[\text{omg t} + x^2 + y^2]\left(\text{et0}\left(\frac{3}{2} + \text{Cos}[\text{omg t} + x^2 + y^2]\right) + \right. \\
& \quad \left.\frac{1}{2}\left(-\text{v0}^2\,(\text{eps} + \text{Cos}[\text{omg t} + x^2 + y^2])^2 - \text{u0}^2\,(\text{eps} + \text{Sin}[\text{omg t} + x^2 + y^2])^2\right)\right)
\end{aligned}
$$

y-component

$$
\begin{aligned}
\text{Out[30]} = {} & \text{omg rho0 v0 Cos}[\text{omg t} + x^2 + y^2]\,(\text{eps} + \text{Cos}[\text{omg t} + x^2 + y^2]) + \\
& 2\,\text{rho0 v0}^3\,\text{y Cos}[\text{omg t} + x^2 + y^2]\,(\text{eps} + \text{Cos}[\text{omg t} + x^2 + y^2])^2 - \\
& \text{omg rho0 v0 Sin}[\text{omg t} + x^2 + y^2]\left(\frac{3}{2} + \text{Sin}[\text{omg t} + x^2 + y^2]\right) + \\
& 2\,\text{rho0 u0 v0 x Cos}[\text{omg t} + x^2 + y^2]\,(\text{eps} + \text{Cos}[\text{omg t} + x^2 + y^2])\,(\text{eps} + \text{Sin}[\text{omg t} + x^2 + y^2]) + \\
& \text{rho0 u0 v0 x Cos}[\text{omg t} + x^2 + y^2]\,(\text{eps} + \text{Cos}[\text{omg t} + x^2 + y^2])\,(3 + 2\,\text{Sin}[\text{omg t} + x^2 + y^2]) - \\
& 2\,\text{rho0 v0}^2\,\text{y}\,(\text{eps} + \text{Cos}[\text{omg t} + x^2 + y^2])\,\text{Sin}[\text{omg t} + x^2 + y^2]\,(3 + 2\,\text{Sin}[\text{omg t} + x^2 + y^2]) - \\
& \text{rho0 u0 v0 x Sin}[\text{omg t} + x^2 + y^2]\,(\text{eps} + \text{Sin}[\text{omg t} + x^2 + y^2])\,(3 + 2\,\text{Sin}[\text{omg t} + x^2 + y^2]) + \\
& \frac{8}{3}\,\text{xmu}\,(2\,\text{v0}\,y^2\,\text{Cos}[\text{omg t} + x^2 + y^2] + (\text{v0} - \text{u0 x y})\,\text{Sin}[\text{omg t} + x^2 + y^2]) + \\
& 2\,\text{xmu}\,(2\,\text{v0}\,x^2\,\text{Cos}[\text{omg t} + x^2 + y^2] + (\text{v0} + 2\,\text{u0 x y})\,\text{Sin}[\text{omg t} + x^2 + y^2]) + \\
& 2\,(-1 + \text{gam})\,\text{rho0 y}\left(\frac{3}{2} + \text{Sin}[\text{omg t} + x^2 + y^2]\right)(\,(-\text{et0} + \text{eps v0}^2)\,\text{Sin}[\text{omg t} + x^2 + y^2] - \\
& \quad \text{Cos}[\text{omg t} + x^2 + y^2]\,(\text{eps u0}^2 + (\text{u0}^2 - \text{v0}^2)\,\text{Sin}[\text{omg t} + x^2 + y^2])) + \\
& 2\,(-1 + \text{gam})\,\text{rho0 y Cos}[\text{omg t} + x^2 + y^2]\left(\text{et0}\left(\frac{3}{2} + \text{Cos}[\text{omg t} + x^2 + y^2]\right) + \right. \\
& \quad \left.\frac{1}{2}\left(-\text{v0}^2\,(\text{eps} + \text{Cos}[\text{omg t} + x^2 + y^2])^2 - \text{u0}^2\,(\text{eps} + \text{Sin}[\text{omg t} + x^2 + y^2])^2\right)\right)
\end{aligned}
$$

Energy equation

$$
\begin{aligned}
\text{Out[33]} = \;& \frac{1}{3}\Big(3\,et0\,omg\,rho0\,\text{Cos}[omg\,t+x^2+y^2]\Big(\frac{3}{2}+\text{Cos}[omg\,t+x^2+y^2]\Big)- \\
& 3\,et0\,omg\,rho0\,\text{Sin}[omg\,t+x^2+y^2]\Big(\frac{3}{2}+\text{Sin}[omg\,t+x^2+y^2]\Big)+ \\
& 6\,v0\,x\,xmu\,\text{Sin}[omg\,t+x^2+y^2]\,(2\,u0\,y\,\text{Cos}[omg\,t+x^2+y^2]-2\,v0\,x\,\text{Sin}[omg\,t+x^2+y^2])-12\,u0 \\
& xmu\,y\,\text{Cos}[omg\,t+x^2+y^2]\,(u0\,y\,\text{Cos}[omg\,t+x^2+y^2]-v0\,x\,\text{Sin}[omg\,t+x^2+y^2])+8\,u0\,xmu\,(\\
& eps+\text{Sin}[omg\,t+x^2+y^2])\,(-(u0+v0\,x\,y)\,\text{Cos}[omg\,t+x^2+y^2]+2\,u0\,x^2\,\text{Sin}[omg\,t+x^2+y^2])- \\
& 8\,u0\,x\,xmu\,\text{Cos}[omg\,t+x^2+y^2]\,(2\,u0\,x\,\text{Cos}[omg\,t+x^2+y^2]+v0\,y\,\text{Sin}[omg\,t+x^2+y^2])- \\
& 8\,v0\,xmu\,y\,\text{Sin}[omg\,t+x^2+y^2]\,(u0\,x\,\text{Cos}[omg\,t+x^2+y^2]+2\,v0\,y\,\text{Sin}[omg\,t+x^2+y^2])+ \\
& 6\,u0\,xmu\,(eps+\text{Sin}[omg\,t+x^2+y^2]) \\
& \quad(-(u0-2\,v0\,x\,y)\,\text{Cos}[omg\,t+x^2+y^2]+2\,u0\,y^2\,\text{Sin}[omg\,t+x^2+y^2])+8\,v0\,xmu\,(\\
& \quad eps+\text{Cos}[omg\,t+x^2+y^2])\,(2\,v0\,y^2\,\text{Cos}[omg\,t+x^2+y^2]+(v0-u0\,x\,y)\,\text{Sin}[omg\,t+x^2+y^2])+ \\
& 6\,v0\,xmu\,(eps+\text{Cos}[omg\,t+x^2+y^2]) \\
& \quad(2\,v0\,x^2\,\text{Cos}[omg\,t+x^2+y^2]+(v0+2\,u0\,x\,y)\,\text{Sin}[omg\,t+x^2+y^2])+ \\
& \frac{1}{gasR}(3\,(-1+gam)\,tcK\,(4\,(u0^2-v0^2)\,x^2\,\text{Cos}[2\,(omg\,t+x^2+y^2)]+ \\
& \quad 2\,(et0-eps\,v0^2-2\,eps\,u0^2\,x^2)\,\text{Sin}[omg\,t+x^2+y^2]+ \\
& \quad 2\,\text{Cos}[omg\,t+x^2+y^2]\,(2\,et0\,x^2+eps\,(u0^2-2\,v0^2\,x^2)+(u0^2-v0^2)\,\text{Sin}[omg\,t+x^2+y^2])))+ \\
& \frac{1}{gasR}(3\,(-1+gam)\,tcK\,(4\,(u0^2-v0^2)\,y^2\,\text{Cos}[2\,(omg\,t+x^2+y^2)]+ \\
& \quad 2\,(et0-eps\,v0^2-2\,eps\,u0^2\,y^2)\,\text{Sin}[omg\,t+x^2+y^2]+ \\
& \quad 2\,\text{Cos}[omg\,t+x^2+y^2]\,(2\,et0\,y^2+eps\,(u0^2-2\,v0^2\,y^2)+(u0^2-v0^2)\,\text{Sin}[omg\,t+x^2+y^2])))+ \\
& 6\,rho0\,u0\,x\,\text{Cos}[omg\,t+x^2+y^2]\Big(\frac{3}{2}+\text{Sin}[omg\,t+x^2+y^2]\Big) \\
& \Big(et0\Big(\frac{3}{2}+\text{Cos}[omg\,t+x^2+y^2]\Big)+(-1+gam)\Big(et0\Big(\frac{3}{2}+\text{Cos}[omg\,t+x^2+y^2]\Big)+ \\
& \quad\frac{1}{2}\left(-v0^2\,(eps+\text{Cos}[omg\,t+x^2+y^2])^2-u0^2\,(eps+\text{Sin}[omg\,t+x^2+y^2])^2\right)\Big)\Big)- \\
& 6\,rho0\,v0\,y\,\text{Sin}[omg\,t+x^2+y^2]\Big(\frac{3}{2}+\text{Sin}[omg\,t+x^2+y^2]\Big) \\
& \Big(et0\Big(\frac{3}{2}+\text{Cos}[omg\,t+x^2+y^2]\Big)+(-1+gam)\Big(et0\Big(\frac{3}{2}+\text{Cos}[omg\,t+x^2+y^2]\Big)+ \\
& \quad\frac{1}{2}\left(-v0^2\,(eps+\text{Cos}[omg\,t+x^2+y^2])^2-u0^2\,(eps+\text{Sin}[omg\,t+x^2+y^2])^2\right)\Big)\Big)+ \\
& 3\,rho0\,v0\,y\,(eps+\text{Cos}[omg\,t+x^2+y^2])\Big(et0\,\text{Cos}[omg\,t+x^2+y^2]\,(3+2\,\text{Cos}[omg\,t+x^2+y^2])- \\
& \quad et0\,\text{Sin}[omg\,t+x^2+y^2]\,(3+2\,\text{Sin}[omg\,t+x^2+y^2])+ \\
& \quad 2\,(-1+gam)\Big(\frac{3}{2}+\text{Sin}[omg\,t+x^2+y^2]\Big)\,((-et0+eps\,v0^2)\,\text{Sin}[omg\,t+x^2+y^2]- \\
& \quad\quad \text{Cos}[omg\,t+x^2+y^2]\,(eps\,u0^2+(u0^2-v0^2)\,\text{Sin}[omg\,t+x^2+y^2]))+ \\
& \quad 2\,(-1+gam)\,\text{Cos}[omg\,t+x^2+y^2]\Big(et0\Big(\frac{3}{2}+\text{Cos}[omg\,t+x^2+y^2]\Big)+ \\
& \quad\quad\frac{1}{2}\left(-v0^2\,(eps+\text{Cos}[omg\,t+x^2+y^2])^2-u0^2\,(eps+\text{Sin}[omg\,t+x^2+y^2])^2\right)\Big)+ \\
& 3\,rho0\,u0\,x\,(eps+\text{Sin}[omg\,t+x^2+y^2])\Big(et0\,\text{Cos}[omg\,t+x^2+y^2]\,(3+2\,\text{Cos}[omg\,t+x^2+y^2])- \\
& \quad et0\,\text{Sin}[omg\,t+x^2+y^2]\,(3+2\,\text{Sin}[omg\,t+x^2+y^2])+ \\
& \quad 2\,(-1+gam)\Big(\frac{3}{2}+\text{Sin}[omg\,t+x^2+y^2]\Big)\,((-et0+eps\,v0^2)\,\text{Sin}[omg\,t+x^2+y^2]- \\
& \quad\quad \text{Cos}[omg\,t+x^2+y^2]\,(eps\,u0^2+(u0^2-v0^2)\,\text{Sin}[omg\,t+x^2+y^2]))+ \\
& \quad 2\,(-1+gam)\,\text{Cos}[omg\,t+x^2+y^2]\Big(et0\Big(\frac{3}{2}+\text{Cos}[omg\,t+x^2+y^2]\Big)+ \\
& \quad\quad\frac{1}{2}\left(-v0^2\,(eps+\text{Cos}[omg\,t+x^2+y^2])^2-u0^2\,(eps+\text{Sin}[omg\,t+x^2+y^2])^2\right)\Big)\Big)\Big)
\end{aligned}
$$

Appendix E: Results of Twenty-one Blind Tests

E.1 Incorrect Array Index

The correct line in NS2D,

```
dvdy(i,j) = ( v(i,j+1) - v(i,j-1) ) *R2Dy
```

was changed to

```
dvdy(i,j) = ( v(i,j) - v(i,j-1) ) *R2Dy
```

The affected part of the above statement is underlined. We expected the grid convergence test to detect this mistake because the approximation to the derivative is not second-order accurate. Table E1 presents relative errors for all the variables using l_2-norm and the maximum error. The order-of-accuracy for all variables have dropped from second to zeroth-order. Therefore, the mistake can be classified as an OAM and is also a *consistency mistake*. This illustrates the grid convergence test can detect a typographically minor error in indexing.

Table E1: Incorrect Array Index

Density						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	9.23708E-4			1.68128E-3		
21x17	8.84707E-4	1.04	0.06	1.73540E-3	0.97	-0.05
41x33	9.44778E-4	0.94	-0.09	1.94904E-3	0.89	-0.17
U-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	6.00723E-4			9.98966E-4		
21x17	3.06844E-4	1.96	0.97	5.75052E-4	1.74	0.80
41x33	2.38584E-4	1.29	0.36	4.96834E-4	1.16	0.21
V-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	9.13941E-3			1.50775E-2		
21x17	9.76774E-3	0.94	-0.10	1.72128E-2	0.88	-0.19
41x33	1.01795E-2	0.96	-0.06	1.80794E-2	0.95	-0.07
Total Energy						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.59926E-4			1.04577E-3		
21x17	1.85468E-4	1.40	0.49	5.29238E-4	1.98	0.98
41x33	2.15114E-4	0.86	-0.21	6.10028E-4	0.87	-0.20

E.2 Duplicate Index

The correct line in NS2D,

```
E(i,j) = Rho_u(i,j) * (Rho_et(i,j) + P(i,j) ) / Rho(i,j)
```

was changed to

```
E(i,j) = Rho_u(i,j) * (Rho_et(i,i) + P(i,j) ) / Rho(i,j)
```

We expected the grid convergence test to find this error because the flux is incorrectly calculated. Table E2 shows the order-of-accuracy for all variables which have dropped from second to the zeroth-order. Thus, the mistake is a *consistency mistake*. This illustrates that convergence testing can detect a typographically minor error such as a duplicate index.

Table E2: Duplicate Index

Density						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.26266E-3			3.93677E-3		
21x17	1.90677E-3	1.19	0.25	3.74222E-3	1.05	0.07
41x33	1.83389E-3	1.04	0.06	3.75482E-3	1.00	0.00
U-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.62133E-3			5.01838E-3		
21x17	2.27151E-3	1.15	0.21	4.78571E-3	1.05	0.07
41x33	2.17877E-3	1.04	0.06	4.83372E-3	0.99	-0.01
V-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	6.62900E-3			2.02920E-2		
21x17	6.47947E-3	1.02	0.03	1.75281E-2	1.16	0.21
41x33	6.54676E-3	0.99	-0.01	1.73238E-2	1.01	0.02
Total Energy						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	6.49276E-2			1.81950E-1		
21x17	6.54015E-2	0.99	-0.01	2.01030E-1	0.91	-0.14
41x33	6.51065E-2	1.00	0.01	2.06200E-1	0.97	-0.04

E.3 Incorrect Constant

The correct line in NS2D,

```
R2Dy = 1.0 / ( 2.0*Dy )
```

was changed to

```
R2Dy = 1.0 / ( 4.0*Dy )
```

We expected the grid convergence test to find this error because it directly affects the order of the approximation. Table E3 presents the error and the order-of-accuracy for all the variables. Here, the error was severe enough that the order-of-accuracy dropped from second to the zeroth-order. Therefore, this is a *consistency mistake*. This illustrates that convergence testing can detect a typographically minor error such as an incorrect constant.

Table E3: Incorrect Constant

Density						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	5.31931E-3			9.79076E-3		
21x17	4.82802E-3	1.10	0.14	9.50070E-3	1.03	0.04
41x33	4.64876E-3	1.04	0.05	9.46390E-3	1.00	0.01
U-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	8.84338E-3			1.48811E-3		
21x17	8.18477E-3	1.08	0.11	1.46418E-3	1.02	0.02
41x33	7.92302E-3	1.03	0.05	1.46074E-3	1.00	0.00
V-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.83176E-2			6.85403E-2		
21x17	3.66663E-2	1.05	0.06	6.90452E-2	0.99	-0.01
41x33	3.58647E-2	1.02	0.03	6.97957E-2	0.99	-0.02
Total Energy						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	6.09470E-3			1.02714E-2		
21x17	5.70251E-3	1.07	0.10	1.02169E-2	1.01	0.01
41x33	5.54180E-3	1.03	0.04	1.02487E-2	1.00	0.00

E.4 Incorrect Do Loop Range

The correct line in NS2D,

```
do i=2,imax-1
```

was changed to

```
do i=2,imax-2
```

We expected the grid convergence test to find this mistake because the density array is not correctly updated. Table E4 shows the behavior of the discretization error on multiple grids and the order-of-accuracy. In this case, the error was severe enough that the solution did not converge for the 41x33 grid. For the remaining grids, the orders of accuracy were less than one which clearly indicates there is something wrong. The affected do loop was used to update the density values. As a result of this change to the range of the loop, part of the computational domain density did not get updated and remained the same as it was initialized throughout the calculation. This is an interesting case, since the outcome of the grid convergence test depends on how the solution is initialized. *If we had initialized the solution to the exact answer our grid convergence test would not have identified the error.* Thus, one should make it a practice to always initialize the solution to something different than the exact answer. This mistake can be classified as an OAM.

Table E4: Incorrect Do Loop Range

Density						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.10730E-1			9.00000E-1		
21x17	2.39100E-1	1.30	0.38	9.00000E-1	1.00	0.00
41x33	Did not Converge					
U-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	4.25453E-2			1.30070E-1		
21x17	3.07927E-2	1.38	0.47	8.75416E-2	1.49	0.57
41x33	Did not Converge					
V-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.37892E-2			1.12420E-1		
21x17	2.50287E-2	1.35	0.43	8.26705E-2	1.36	0.44
41x33	Did not Converge					
Total Energy						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.55018E-2			1.10310E-1		
21x17	2.44229E-2	1.45	0.54	6.72695E-2	1.64	0.71
41x33	Did not Converge					

E.5 Uninitialized Variable

The correct line in NS2D,

```
f23 = 2.0 / 3.0
```

was changed to a comment

```
c       f23 = 2.0 / 3.0
```

This type of mistake should have been detected by the static test. Most compilers would issue a warning that a variable is used before being initialized. Also, all the FORTRAN checkers would detect this mistake. Let's assume that the warning was ignored and see what happens when we run the grid convergence test. We expected the grid convergence test to find this mistake and it did. The mistake was severe enough that the code did not converge on the 41x33 grid. For the remaining grids, as shown in Table E5, the order-of-accuracy for all variables was about zero. Thus, the grid convergence test can detect mistakes such as uninitialized variables. This mistake is properly classified as a *static mistake* and illustrates that MMS can sometimes detect coding mistakes that are not OAM's.

Table E5: Uninitialized Variable

Density

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.65560E-2			5.38971E-2		
21x17	2.52451E-2	1.05	0.07	5.58692E-2	0.96	-0.05
41x33	Did not Converge					

U-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.01007E-2			5.83826E-2		
21x17	2.90318E-2	1.04	0.05	6.07713E-2	0.96	-0.06
41x33	Did not Converge					

V-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	5.24298E-3			1.50089E-2		
21x17	4.26897E-3	1.23	0.30	1.16589E-2	1.29	0.36
41x33	Did not Converge					

Total Energy

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.55018E-2			1.48726E-2		
21x17	2.44229E-2	1.45	0.54	1.77831E-2	0.84	-0.26
41x33	Did not Converge					

E.6 Incorrect Labeling of an Array in an Argument List

The correct line in NS2D,

```
dudx,dvdx,dtdx, dudy,dvdy,dtdy,
```

was changed to

```
dudx,dudy,dtdx, dvdx,dvdy,dtdy,
```

The above line appears on the calling statement to the routine that solves the energy equation. The mistake interchanged the arrays that hold the $\partial u/\partial y$ and $\partial v/\partial x$ derivatives. We expected the grid convergence test to find this error and it did NOT! Table E6 shows that the order-of-accuracy for all variables to be second-order accurate. This was surprising but, after studying the problem, it was found that the derivatives $\partial u/\partial y$ and $\partial v/\partial x$ are only used in constructing a component of shear stress tensor τ_{xy} which is defined as

$$\tau_{xy} = \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \tag{50}$$

In Equation 50, the two derivatives are added together and thus, it did not matter which array holds what derivative. The grid convergence test did not detect this mistake because the solution was not affected. This mistake falls in the category of a *formal coding mistake*.

Table E6: Incorrect Labeling of an Array in an Argument List

Density

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.91885E-4			8.33373E-4		
21x17	9.94661E-5	3.94	1.98	1.99027E-4	4.19	2.07
41x33	2.50657E-5	3.97	1.99	4.93822E-5	4.03	2.01

U-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.31435E-4			5.98882E-4		
21x17	7.97201E-5	4.16	2.06	1.50393E-4	3.98	1.99
41x33	1.98561E-5	4.01	2.01	3.76816E-5	3.99	2.00

V-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.28458E-4			3.83091E-4		
21x17	4.87140E-5	4.69	2.23	8.58549E-5	4.46	2.16
41x33	1.13447E-5	4.29	2.10	2.09721E-5	4.09	2.03

Total Energy

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.90889E-4			3.51916E-4		
21x17	4.20472E-5	4.54	2.18	8.86208E-5	3.97	1.99
41x33	1.00610E-5	4.18	2.06	2.18304E-5	4.06	2.02

E.7 Switching of the Inner and the Outer Loop Indices

The correct line in NS2D,

```
j=2,jmax-1
  i=2,imax-1
```

was changed to

```
i=2,jmax-1
  j=2,imax-1
```

This mistake would have been detected by the dynamic test of the code with the array bound checker turned on. In the case of dynamically allocated memory, the code will try to access memory locations that are not defined for the individual arrays in the do loop which typically causes the code to terminate. Lets assume the code has declared all arrays big enough that the code would run. *We expected the grid convergence test to find this mistake when jmax does not equal imax and it did.* Table E7 shows the results for the grid convergence test. The mistake was severe enough that the code did not converge for the second and the third grids of the test. This mistake falls in our taxonomy under the category of a *divergence mistake.*

Table E7: Switching of the Inner and the Outer Loop Indices

Density

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.75040E-1			6.80710E-1		
21x17			Did not Converge			
41x33						

U-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.01780E-1			3.01180E-1		
21x17			Did not Converge			
41x33						

V-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.94940E-1			1.77269E-1		
21x17			Did not converge			
41x33						

Total Energy

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.07015E-1			1.04740E-1		
21x17			Did not Converge			
41x33						

E.8 Incorrect Sign

The correct line in NS2D,

```
P_new = (gam-1)*Rho(i,j)*( et - 0.5*(u*u+v*v) )
```

was changed to

```
P_new = (gam+1)*Rho(i,j)*( et - 0.5*(u*u+v*v) )
```

We expected the grid convergence test to find this mistake and it did. Table E8 shows that the l_2-norm of the error is converging first-order and the max error shows near zeroth-order convergence. This illustrates that grid convergence tests can detect a typographically minor mistake such as an incorrect sign. Also, this mistake is obvious enough that it would have likely been caught during the development stage. This mistake can be classified as an OAM.

Table E8: Incorrect Sign

Density

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	5.24657E-4			1.20947E-3		
21x17	4.35951E-4	1.20	0.27	1.04079E-3	1.16	0.22
41x33	3.78324E-4	1.15	0.20	8.68705E-4	1.20	0.26

U-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	4.49409E-4			1.00287E-3		
21x17	4.60418E-4	0.98	-0.03	9.57168E-4	1.05	0.07
41x33	3.60157E-4	1.28	0.35	8.30955E-4	1.15	0.20

V-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.27151E-3			4.69045E-3		
21x17	1.72803E-3	1.31	0.39	3.98002E-3	1.18	0.24
41x33	1.56727E-3	1.10	0.14	3.71797E-3	1.07	0.10

Total Energy

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	4.75690E-1			6.63490E-1		
21x17	4.74390E-1	1.00	0.00	6.79370E-1	0.98	-0.03
41x33	4.73770E-1	1.00	0.00	6.86590E-1	0.99	-0.02

E.9 Incorrect Positioning of Operators

The correct line in NS2D,

```
F(i,j) = Rho_u(i,j)*Rho_v(i,j)/Rho(i,j)
```

was changed to

```
F(i,j) = Rho_u(i,j)/Rho_v(i,j)*Rho(i,j)
```

We expected the grid convergence test to find this mistake and it did. Table E9 shows that the order-of-accuracy for all variables have dropped to zero. The mistake is classified as a *consistency mistake*. Most likely this mistake would have been detected during the code development stage.

Table E9: Incorrect Positioning of Operators

Density						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.01690E-1			3.20620E-1		
21x17	1.95680E-1	1.03	0.04	3.24560E-1	0.99	-0.02
41x33	1.91960E-1	1.02	0.03	3.25250E-1	1.00	0.00
U-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.84100E-1			2.71070E-1		
21x17	1.77910E-1	1.03	0.05	2.76040E-1	0.98	-0.03
41x33	1.74350E-1	1.02	0.03	2.77870E-1	0.99	-0.01
V-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	4.92486E-2			9.41443E-2		
21x17	4.74185E-2	1.04	0.05	9.28039E-2	1.01	0.02
41x33	4.67834E-2	1.01	0.02	9.30090E-2	1.00	0.00
Total Energy						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	5.08252E-2			1.18530E-1		
21x17	4.80429E-2	1.06	0.08	1.04160E-1	1.14	0.19
41x33	4.80885E-2	1.00	0.00	1.03480E-1	1.01	0.01

E.10 Incorrect Parenthesis Position

The correct line in NS2D,

```
dtdy(i,j) = ( -3.0*T(i,j) + 4.0*T(i,j+1) - T(i,j+2) )*R2Dy
```

was changed to

```
dtdy(i,j) = ( -3.0*T(i,j) + 4.0*T(i,j+1) ) - T(i,j+2) *R2Dy
```

We initially expected the grid convergence test to find this mistake and it did NOT! Table E10 shows the second-order accuracy behavior for all the variables. This was surprising but, after careful examination, we found that the above statement was used to compute the temperature gradient at a corner point of the computational grid and the stencil used by the code did not reach the corner points. Therefore, the mistake did not alter the results. However, if this point was inside the computational domain, it would have been detected. This mistake is classified as a *formal mistake*.

Table E10: Incorrect Parenthesis Position

Density						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.91885E-4			8.33373E-4		
21x17	9.94661E-5	3.94	1.98	1.99027E-4	4.19	2.07
41x33	2.50657E-5	3.97	1.99	4.93822E-5	4.03	2.01
U-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.31435E-4			5.98882E-4		
21x17	7.97201E-5	4.16	2.06	1.50393E-4	3.98	1.99
41x33	1.98561E-5	4.01	2.01	3.76816E-5	3.99	2.00
V-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.28458E-4			3.83091E-4		
21x17	4.87140E-5	4.69	2.23	8.58549E-5	4.46	2.16
41x33	1.13447E-5	4.29	2.10	2.09721E-5	4.09	2.03
Total Energy						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.90889E-4			3.51916E-4		
21x17	4.20472E-5	4.54	2.18	8.86208E-5	3.97	1.99
41x33	1.00610E-5	4.18	2.06	2.18304E-5	4.06	2.02

E.11 Conceptual or Consistency Mistake in Differencing Scheme

The correct line in NS2D,

```
-d0*( Rho_u(i+1,j) - Rho_u(i-1,j) )
```

was changed to

```
-d0*( Rho_u(i+1,j) - Rho_u(i-1,j) + Rho_u(i,j) )
```

We expected the grid convergence test to find this mistake and it did. The order-of-accuracy for all variables dropped to zeroth-order as shown in Table E11. This mistake can be classified as either a *dynamic conceptual mistake or a consistency mistake.*

Table E11: Conceptual Error in Differencing Scheme

Density

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	5.59550E-1			7.53550E-1		
21x17	6.76410E-1	0.83	-0.27	8.92800E-1	0.84	-0.24
41x33	7.68920E-1	0.88	-0.18	9.59010E-1	0.93	-0.10

U-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.42**692**E-2			5.94497E-2		
21x17	3.55477E-2	0.96	-0.05	6.66460E-2	0.89	-0.16
41x33	3.49851E-2	1.02	0.02	6.55673E-2	1.02	0.02

V-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.62034E-2			6.39796E-2		
21x17	3.84234E-2	0.94	-0.09	6.60947E-2	0.97	-0.05
41x33	3.91125E-2	0.98	-0.03	6.43425E-2	1.03	0.04

Total Energy

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.53229E-2			4.67058E-2		
21x17	2.53935E-2	1.00	0.00	5.18047E-2	0.90	-0.15
41x33	2.36851E-2	1.07	0.10	5.01506E-2	1.03	0.05

E.12 Logical IF Mistake

The correct line in NS2D,

```
if( sum_Rho_et.le.tol .and. sum_Rho_u.le.tol .and.
    sum_Rho_v .le.tol .and. sum_Rho  .le.tol  )
                converged = .true.
```

was changed to

```
if( sum_Rho_et.ge.tol .and. sum_Rho_u.le.tol .and.
    sum_Rho_v .le.tol .and. sum_Rho  .le.tol  )
                converged = .true.
```

The convergence test for the energy equation was affected by this change. In this particular implementation, the convergence of all the computed variables were tested. This suggests if the convergence tolerance is very small, let say close to the machine zero, then, *the grid convergence test would NOT find this error and it did NOT*. The reason is simple, while there is a faulty convergence check on the total energy, we still check the convergence of the other variables. Since this a coupled systems of equations, when the other variables are converged to a small tolerance then the energy equation most likely has converged within an order of magnitude of the same tolerance. Table E12 shows the order-of-accuracy are unaffected by this mistake. This mistake is classified as a *formal mistake* because we cannot conceive a dynamic test that would reveal this mistake (at least on this particular problem).

Table E12: Logical If Error

Density						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.91885E-4			8.33374E-4		
21x17	9.94661E-5	3.94	1.98	1.99027E-4	4.19	2.07
41x33	2.50656E-5	3.97	1.99	4.93819E-5	4.03	2.01
U-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.31435E-4			5.98882E-4		
21x17	7.97201E-5	4.16	2.06	1.50393E-4	3.98	1.99
41x33	1.98561E-5	4.01	2.01	3.76816E-5	3.99	2.00
V-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.28458E-4			3.83091E-4		
21x17	4.87140E-5	4.69	2.23	8.58549E-5	4.46	2.16
41x33	1.13447E-5	4.29	2.10	2.09720E-5	4.09	2.03
Total Energy						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.90889E-4			3.51916E-4		
21x17	4.20472E-5	4.54	2.18	8.86208E-5	3.97	1.99
41x33	1.00610E-5	4.18	2.06	2.18304E-5	4.06	2.02

E.13 No Mistake

There are no mistakes in case 8.13 (a placebo). This is identical to the original code and it was put in into the test matrix as a check. As we mentioned earlier, this was a blind test. Table E13 shows the second-order accurate convergence behavior for all the variables.

Table E13: No Error

Density						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.91885E-4			8.33373E-4		
21x17	9.94661E-5	3.94	1.98	1.99027E-4	4.19	2.07
41x33	2.50657E-5	3.97	1.99	4.93822E-5	4.03	2.01
U-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.31435E-4			5.98882E-4		
21x17	7.97201E-5	4.16	2.06	1.50393E-4	3.98	1.99
41x33	1.98561E-5	4.01	2.01	3.76816E-5	3.99	2.00
V-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.28458E-4			3.83091E-4		
21x17	4.87140E-5	4.69	2.23	8.58549E-5	4.46	2.16
41x33	1.13447E-5	4.29	2.10	4.09720E-5	4.09	2.03
Total Energy						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.90889E-4			3.51916E-4		
21x17	4.20472E-5	4.54	2.18	8.86208E-5	3.97	1.99
41x33	1.00610E-5	4.18	2.06	2.18304E-5	4.06	2.02

E.14 Incorrect Relaxation Factor

The correct line in NS2D,

```
Rho(i,j) = Rho(i,j) + 0.8*( Rho_new - Rho(i,j) )
```

was changed to

```
Rho(i,j) = Rho(i,j) + 0.6*( Rho_new - Rho(i,j) )
```

We expected the grid convergence test to NOT find this mistake and it did NOT. Table E14 shows second-order accurate convergence for all variables. This illustrates that the grid convergence test will not detect mistakes in the iterative solver which affect only the rate of convergence, but not the accuracy of the answer. This is classified as an *efficiency mistake.*

Table E14: Incorrect Relaxation Factor

Density

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.91885E-4			8.33373E-4		
21x17	9.94656E-5	3.94	1.98	1.99026E-4	4.19	2.07
41x33	2.50638E-5	3.97	1.99	4.93784E-5	4.03	2.01

U-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.31435E-4			5.98882E-4		
21x17	7.97200E-5	4.16	2.06	1.50393E-4	3.98	1.99
41x33	1.98558E-5	4.01	2.01	3.76810E-5	3.99	2.00

V-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.28458E-4			3.83091E-4		
21x17	4.87140E-5	4.69	2.23	8.58550E-5	4.46	2.16
41x33	1.13448E-5	4.29	2.10	2.09722E-5	4.09	2.03

Total Energy

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.90889E-4			3.51916E-4		
21x17	4.20473E-5	4.54	2.18	8.86209E-5	3.97	1.99
41x33	1.00630E-5	4.18	2.06	2.18309E-5	4.06	2.02

E.15 Incorrect Differencing

The correct line in NS2D

```
dtdy(i,j) = ( -3.0*T(i,j) + 4.0*T(i,j+1) - T(i,j+2) )*R2Dy
```

was changed to

```
dtdy(i,j) = ( -3.0*T(i,j) + 4.0*T(i,j+1) - T(i,j+2) )*R2Dx
```

We expected the grid convergence test to find this mistake and it did. Table E15 shows that the energy and v-component of velocity variables exhibit first-order accuracy. This clearly indicates the sensitivity of the MMS procedure. The mistake is classified as an OAM.

Table E15: Incorrect Differencing

Density

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	4.65492E-4			1.00034E-3		
21x17	1.33118E-4	3.50	1.81	2.78774E-4	3.59	1.84
41x33	4.13695E-5	3.22	1.69	9.01491E-5	3.09	1.63

U-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.43574E-4			6.08897E-4		
21x17	8.53543E-5	4.03	2.01	1.62447E-4	3.75	1.91
41x33	2.28791E-5	3.73	1.90	4.77928E-5	3.40	1.77

V-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	6.33114E-4			1.36905E-3		
21x17	3.25274E-4	1.95	0.96	7.73432E-4	1.77	0.82
41x33	1.61930E-4	2.01	1.01	3.99318E-4	1.94	0.95

Total Energy

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.57412E-3			6.63811E-3		
21x17	1.19438E-3	2.16	1.11	3.58674E-3	1.85	0.89
41x33	5.70262E-4	2.09	1.07	1.86694E-3	1.92	0.94

E.16 Missing Term

The correct line in NS2D,

```
E(i,j) = Rho_u(i,j)*( Rho_et(i,j)+P(i,j) )/Rho(i,j)
```

was changed to

```
E(i,j) = Rho_u(i,j)*( Rho_et(i,j) )/Rho(i,j)
```

We expected grid convergence test to find this mistake and it did. Table E16 shows that the order-of-accuracy for all variables have dropped to zero. This illustrates grid convergence test can find a missing term. This is classified as a *consistency mistake*.

Table E16: Missing Term

Density

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.94477E-4			7.68192E-4		
21x17	1.07306E-4	2.74	1.46	2.93401E-4	2.62	1.39
41x33	1.25625E-4	0.85	-0.23	2.68408E-4	1.09	0.13

U-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.01106E-4			3.32522E-4		
21x17	6.36136E-5	3.16	1.66	1.56022E-4	2.13	1.09
41x33	1.13422E-4	0.56	-0.83	2.48116E-4	0.63	-0.67

V-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	7.75258E-4			1.53890E-3		
21x17	6.17087E-4	1.26	0.33	1.39906E-3	1.10	0.14
41x33	5.82296E-4	1.06	0.08	1.36553E-3	1.02	0.03

Total Energy

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.98107E-3			8.56773E-3		
21x17	3.98790E-3	1.00	0.00	9.36147E-3	0.92	-0.13
41x33	3.95471E-3	1.01	0.01	9.63853E-3	0.97	-0.04

E.17 Distortion of a Grid Point

An extra line was added to NS2D to distort the grid:

```
x(1,1) = x(1,1) - 0.25*Dx
```

We expected the grid convergence test to find this mistake because the code requires a Cartesian mesh. Because the code contains its own grid generator, this mistake is not an input violation. Table E17 shows the error and the convergence behavior for all the variables. The order-of-accuracy based on the l_2-norm shows second-order convergence. This would indicate that there are no mistakes in the code. However, the order-of-accuracy computed based on the maximum error clearly points to the convergence of the v-component of velocity which has dropped to first-order. This was a good exercise that shows that the maximum error is more sensitive than the l_2-norm of the error. This mistake is classified as an OAM

Table E17: Distortion of a Grid Point

Density						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.94756E-4			8.34218E-4		
21x17	9.99209E-5	3.95	1.98	1.99131E-4	4.19	2.07
41x33	2.51496E-5	3.97	1.99	5.34937E-5	3.72	1.90
U-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.32636E-4			6.04219E-4		
21x17	7.99316E-5	4.16	2.06	1.50968E-4	4.00	2.00
41x33	1.98889E-5	4.02	2.01	3.77957E-5	3.99	2.00
V-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.80813E-4			1.26750E-3		
21x17	6.07246E-5	4.62	2.21	5.81576E-4	2.18	1.12
41x33	1.42494E-5	4.26	2.09	2.84931E-4	2.04	1.03
Total Energy						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.91004E-4			3.51126E-4		
21x17	4.20463E-5	4.54	2.18	8.85215E-5	3.97	1.99
41x33	1.00597E-5	4.18	2.06	2.18118E-5	4.06	2.02

E.18 Incorrect Position of an Operator in Output Calculation

The correct line in NS2D,

```
Rho_u(i,j)/Rho(i,j), Rho_v(i,j)/Rho(i,j),
```

in the velocity output was change to

```
Rho_u(i,j)*Rho(i,j), Rho_v(i,j)/Rho(i,j),
```

Since we have computed the errors inside the code as opposed to using the output of the code, we did NOT expect the grid convergence test to find this error and it did NOT. Table E18 shows proper convergence behavior for all the variables. *We recommend that all discretization errors to be computed using the output of the code;* this will result in detection of mistakes in the output routines. If the output had been used to compute the discretization error, this mistake would be classified as an OAM. Since the output was not used, the mistake can be classified as a *formal error*.

Table E18: Incorrect Position of an Operator

Density						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.91885E-4			8.33373E-4		
21x17	9.94661E-5	3.94	1.98	1.99027E-4	4.19	2.07
41x33	2.50657E-5	3.97	1.99	4.93822E-5	4.03	2.01
U-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.31435E-4			5.98882E-4		
21x17	7.97201E-5	4.16	2.06	1.50393E-4	3.98	1.99
41x33	1.98561E-5	4.01	2.01	3.76816E-5	3.99	2.00
V-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.28458E-4			3.83091E-4		
21x17	4.87140E-5	4.69	2.23	8.58549E-5	4.46	2.16
41x33	1.13447E-5	4.29	2.10	2.09721E-5	4.09	2.03
Total Energy						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.90889E-4			3.51916E-4		
21x17	4.20472E-5	4.54	2.18	8.86208E-5	3.97	1.99
41x33	1.00610E-5	4.18	2.06	2.18304E-5	4.06	2.02

E.19 Change the number of elements in the grid

The following lines

```
imax = imax + 1
jmax = jmax + 1
```

were added to NS2D after the statement that defines the problem size. *We did NOT expect the grid convergence test to find this mistake and it did NOT.* This mistake modified the size of the grid by one in each direction which in turn affects the grid refinement ratio. The influence on this ratio is more pronounced on the coarse grid and starts to go down as we refine the mesh. Table E19 shows the order-of-accuracy for all variables. We decided to do one additional level of refinement since we were concerned about the magnitude of the order of convergence. With the added level, we concluded from the grid convergence results that there were no mistakes in the code. This mistake can only be caught if one uses or examines the code output (which we did not do in this example). For the purpose of this exercise, we classify this mistake as a *formal mistake*.

Table E19: Change the Number of Elements in the Grid

Density						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.21339E-4			6.98219E-4		
21x17	8.98044E-5	3.58	1.84	1.81494E-4	3.85	1.94
41x33	2.37936E-5	3.77	1.92	4.70610E-5	3.86	1.95
81x65	6.11793E-6	3.89	1.96	1.19481E-5	3.94	1.98
U-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.66476E-4			4.98390E-4		
21x17	7.16769E-5	3.72	1.89	1.37977E-4	3.61	1.85
41x33	1.88277E-5	3.81	1.93	3.61425E-5	3.82	1.93
81x65	4.84251E-6	3.89	1.96	9.19734E-6	3.93	1.97
V-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.77446E-4			3.05520E-4		
21x17	4.32666E-5	4.10	2.04	7.79543E-5	3.92	1.97
41x33	1.07127E-5	4.04	2.01	1.97842E-5	3.94	1.98
81x65	2.66420E-6	4.02	2.01	5.00060E-6	3.96	1.98
Total Energy						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.51588E-4			2.81122E-4		
21x17	3.77868E-5	4.01	2.00	7.89060E-5	3.56	1.83
41x33	9.54920E-6	3.96	1.98	2.06421E-5	3.82	1.93
81x65	2.40351E-6	3.97	1.99	5.21747E-6	3.96	1.98

E.20 Redundant Do Loop

A redundant do loop:

```
do j=2,jmax-1
   do i=2,imax-1
      dudx(i,j) = ( u(i+1,j) - u(i-1,j) ) *R2Dx
       ......
       ......
   enddo
enddo
```

was added to the NS2D. *We did NOT expect the grid convergence test to detect this mistake and it did NOT.* Table E20 shows second-order-of-accuracy for all the variables. Because this mistake might, in principle, be detected by code performance testing, we classify it as an *efficiency mistake*, although one could argue also that it is a *formal error*.

Table E20: Redundant Do Loop

Density

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.91885E-4			8.33373E-4		
21x17	9.94661E-5	3.94	1.98	1.99027E-4	4.19	2.07
41x33	2.50657E-5	3.97	1.99	4.93822E-5	4.03	2.01

U-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.31435E-4			5.98882E-4		
21x17	7.97201E-5	4.16	2.06	1.50393E-4	3.98	1.99
41x33	1.98561E-5	4.01	2.01	3.76816E-5	3.99	2.00

V-Component of Velocity

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.28458E-4			3.83091E-4		
21x17	4.87140E-5	4.69	2.23	8.58549E-5	4.46	2.16
41x33	1.13447E-5	4.29	2.10	2.09720E-5	4.09	2.03

Total Energy

Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.90889E-4			3.51916E-4		
21x17	4.20472E-5	4.54	2.18	8.86208E-5	3.97	1.99
41x33	1.00610E-5	4.18	2.06	2.18304E-5	4.06	2.02

E.21 Incorrect Value of the Time Step

The value of the time-step Δt was changed to $0.8*\Delta t$ in the NS2D. As a result, the code was running with slightly smaller time step. *For steady-state problems, we expected the grid convergence test to NOT find this mistake and it did NOT.* Since getting to the steady state solution is not dependent on Δt (this is not always true), the solution is unaffected by this mistake. Table E21 shows the correct order-of-accuracy for all the variables. However, the same mistake would have been detected if we were solving an unsteady problem. This mistake is classified as a *formal mistake.*

Table E21: Incorrect Value of Time-Step

Density						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.91885E-4			8.33373E-4		
21x17	9.94658E-5	3.94	1.98	1.99026E-4	4.19	2.07
41x33	2.50647E-5	3.97	1.99	4.93798E-5	4.03	2.01
U-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	3.31435E-4			5.98882E-4		
21x17	7.97201E-5	4.16	2.06	1.50393E-4	3.98	1.99
41x33	1.98561E-5	4.01	2.01	3.76817E-5	3.99	2.00
V-Component of Velocity						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	2.28458E-4			3.83091E-4		
21x17	4.87140E-5	4.69	2.23	8.58548E-5	4.46	2.16
41x33	1.13446E-5	4.29	2.10	2.09717E-5	4.09	2.03
Total Energy						
Grid	l_2-norm	Ratio	Observed Order	Max error	Ratio	Observed Order
11x9	1.90889E-4			3.51916E-4		
21x17	4.20472E-5	4.54	2.18	8.86207E-5	3.97	1.99
41x33	1.00610E-5	4.18	2.06	2.18301E-5	4.06	2.02

External Distribution

Adams, M. A.
Jet Propulsion Laboratory
4800 Oak Grove Drive, MS 97
Pasadena, CA 91109

Anderson, Charles E.
Southwest Research Institute
P.O. Drawer 28510
San Antonio, TX 78284

Anderson, D.
Graduate Aernonautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 205-45
Pasadena, CA 91125

Anderson, J. D.
Smithsonian Institute,
National Air and Space Museum
1800 Billman Lane
Silver Spring, MD 20902

Aivasis, M.
Center for Advanced Computing Research
California Institute of Technology
1200 E. California Blvd./MS 158-79
Pasadena, CA 91125

Anderson, M.
ACTA, 2790 Skypark Dr., Suite 310
Torrance, CA 90505-5345

Ayyub, B. M
University of Maryland
Civil Engineering - Rm. 1155
College Park, MD 20742-3021

Beissel, S.
Alliant Techsystems Inc.
600 Second St. NE
Hopkins, MN 55343

Belytschko, T.
Northwestern University
Dept. of Mechanical Engineering
2145 Sheridan Road
Evanston, IL 60108-3111

Cafeo, John A.
General Motors
Research & Development Center
Mail Code 480-106-256
30500 Mound Road, Box 9055
Warren, MI 48090-9055

Casteel, K.
University of Texas - El Paso
600 W. University Ave
El Paso, TX 79968-0521

Cavendish, James C.
General Motors
Research & Development Center
Mail Code 480-106-359
30500 Mound Road, Box 9055
Warren, MI 48090-9055

Christon, M. A.
Livermore Software Technology Corporation
7209 Aztec Rd., NE
Albuquerque, NM 87110

Chwastyk,T.
U.S. Naval Research Laboratory
Code 6304, 4555 Overlook Avenue SW
Washington DC 20375-5343

Cho, K.
Stanford University
Durand 267
Stanford, CA 94305

Colemans, H. W.
U. Alabama Huntsville
S236 Technology Hall
Huntsville, AL 35899

Coker, D.
Graduate Aeronautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 105-50
Pasadena, CA 91125

Cosner, R. R.
Senior Technical Fellow
Boeing - Phantom Work
P. O. Box 516, MS: S106 - 7126
St. Louis, MO 63166

Cuniff, P.
U. S. Army Soldier Systems Center
Kansas Street
Natick, MA 01750-5019

Diwekar, U. M.
Center for Energy and Environmental Studies
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Gabrovsek, R.
National Institute of Chemistry
Hajdrihova 19, POB 3430
SI-1000 Ljubljana
Slovenia

Glimm, J. G.
Dept. of Applied Math and Statistics
Math, P138A
State University of New York at Stony Brook
Stony Brook, NY 11794-3600

Gnoffo, P. A.
NASA Langley Research Center
Aerothermodynamics Branch
Mail Stop 408A
Hampton, VA 23681-0001

Hasselman, T. K.
ACTA
2790 Skypark Dr., Suite 310
Torrance, CA 90505-5345

Hills, R. G.
Department of Mechanical Engineering
New Mexico State University
Las Cruces, New Mexico 88001

Howes, F. A.
U. S. Dept. of Energy
DOE, MICS, ER-31
19901 Germantown Rd.
Germantown, MD 20874

Ivy, G.
Logicon R&D Associates
P. O. Box 92500
Los Angeles, CA 90009

Kane, C.
California Institute of Technology
1200 E. California Blvd./MS 205-45
Pasadena, CA 91125

Karniadakis, G.
Division of Applied Mathematics
Brown University
192 George St., Box F
Providence, RI 02912

Keremes, J.
The Boeing Company
Rocketdyne Propulsion & Power
6633 Canoga Avenue
Canoga Park, CA 91309-7922

Kimsey, K. D.
U. S. Army Research Laboratory
Weapons and Materials Research
Directorate
AMSRL-WM-TC 309 120A
Aberdeen Proving Ground, MD 21005-5066

Kovac, B. A.
The Boeing Company
Rocketdyne Propulsion and Power
6633 Canoga Avenue
Canoga Park, CA 91309-7922

Krysl, P.
Department of Computer Science
California Institute of Technology
1200 E. California Blvd./MS 256-80
Pasadena, CA 91125

Leonard, T.
California Institute of Technology
1200 East California Blvd. M/C 301-46
Pasadena, CA 91125

Lin, T. C.
TRW Ballistic Missile Division
Bldg. 953, Room 2340
P.O. Box 1310
San Bernadino, CA 92402-1310

Liu, W. K.
Northwestern University
Dept. of Mechanical Engineering
2145 Sheridan Road
Evanston, IL 60108-3111

Mair, H. U.
Institute of Defense Analyses
Operational Evaluation Division
1801 North Beauregard Street
Alexandria, VA 22311-1772

McRae, G.
Department of Chemical Engineering
Massachusetts Institute of Technology
Cambridge, MA 02139

McDonald, W.
Naval Surface Warfare Center
Code 420
101 Strauss Avenue
Indian Head, MD 20640-5035

Mehta, U. B.
NASA Ames Research Center
MS: T27 B-1
Moffett Field, CA 94035-1000

Molinari, J.-F.
Graduate Aernonautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 105-50
Pasadena, CA 91125

Morris, M. D.
Oak Ridge National Lab
P. O. Box 2008
Building 6012
Oak Ridge, TN 37831-6367

Namburu, R.
Director
U.S. Army Research Laboratory
Attn: AMSRL-CI-H (Dr. Raju Namburu)
APG, MD 21005-5067

Needham, C.
Applied Research Associates, Inc.
4300 San Mateo Blvd. Ste A-220
Albuquerque, NM 87110

Needleman, A.
Brown University
Division of Engineering, Box D
Providence, RI 02912

Omprakash, S.
Graduate Aeronautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 105-50
Pasadena, CA 91125

Ortiz, M.
Graduate Aeronautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 105-50
Pasadena, CA 91125

Pace, D. K.
The John Hopkins University
Applied Physics Laboratory
111000 Johns Hopkins Road
Laurel, MD 20723-6099

Papoulia, K.
Inst. Eng. Seismology & Earthquake Engineering
P. O. Box 53, Finikas GR-55105
Thessaloniki, Greece

Parrish, D. K.
Centerline Consulting
840 N. Spruce St., Unit 308
Rapid City, SD 57701

Radovitzky, P.
Graduate Aeronautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 105-50
Pasadena, CA 91125

Rafaniello, W.
DOW Chemical Company
1776 Building
Midland, MI 48674

Raj, P.
Lockheed Martin Aeronautical Systems
D/73-07 Z/0685
86 S. Cobb Drive
Marietta, GA 30063-1000

Ran, H.
Graduate Aeronautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 105-50
Pasadena, CA 91125

Repetto, E.
Graduate Aeronautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 105-50
Pasadena, CA 91125

Roache, P. J. (3)
Hermosa Publishers
P.O. Box 9110
Albuquerque, NM 87119-9110

Rosakis, A. J.
Graduate Aeronautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 105-50
Pasadena, CA 91125

Sevin, E.
Logicon RDA, Inc.
1782 Kenton Circle
Lyndhurst, OH 44124

Shu, Y. C.
Graduate Aeronautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 105-50
Pasadena, CA 91125

Singleton, R.
Director, Engineering Sciences Directorate
U. S. Army Research Office
4300 S. Miami Blvd.
P. O. Box 1221
Research Triangle Park, NC 27709-2211

Snowden, W. E.
DARPA
7120 Laketree Drive
Fairfax Station, VA 22039

Stevenson, D. E. (Steve)
Clemson University
Computer Science Department
442 Edwards Hall - Box 341906
Clemson, SC 29631-1906

Steinberg, S.
Department of Mathematics and Statistics
University of New Mexico
Albuquerque, NM 87131

Sundaram, S.
Graduate Aeronautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 105-50
Pasadena, CA 91125

Thoutireddy, P.
Graduate Aeronautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 105-50
Pasadena, CA 91125

Truman, C. R.
Mechanical Engineering Department
University of New Mexico
Albuquerque, NM 87131

Voelkl, T.
Graduate Aeronautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 105-50
Pasadena, CA 91125

Wendl, M. C.
Washington University Medical School
4444 Forest Park Blvd., Box 8501
St. Louis, MO 63108

Williamson, W. E.
Department of Engineering
Texas Christian University
Box 298640
Fort Worth, Texas 76129

Xu, L.
Graduate Aeronautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 105-50
Pasadena, CA 91125

Yu, C.
Graduate Aeronautical Laboratories
California Institute of Technology
1200 E. California Blvd./MS 105-50
Pasadena, CA 91125

Zikry, M. A.
North Carolina State University
Mechanical & Aerospace Engineering
2412 Broughton Hall, Box 7910
Raleigh, NC 27695

Zwissler, J. G.
Jet Propulsion Laboratory
4800 Oak Grove Drive - MS 97-8
Pasadena, CA 91109-8099

Los Alamos National Laboratory

Mail Station 5000
P.O. Box 1663
Los Alamos, NM 87545
Attn: D. Cagliostro, MS F645
Attn: D. L. Crane, MS P946
Attn: J. F. Davis, MS P234
Attn: K. M. Hanson, MS P940
Attn: R. Henninger, MS D413
Attn: B. L. Holian, MS B268
Attn: K. S. Holian, MS D413
Attn: J. M. Hyman, MS B284
Attn: M. E. Jones, MS B259
Attn: J. R. Kamm, MS D413
Attn: E. J. Kelly, MS F600
Attn: J. V. Lagrange, MS D445
Attn: L. G. Margolin, MS D413
Attn: M. McKay, MS F600
Attn: M. R. Miller, MS D471
Attn: M.-M. Peterson, CIC-12
Attn: W. J. Rider, MS D413
Attn: D. Sharp, MS B213
Attn: R. N. Silver, MS B262
Attn: D. Weeks, MS B295

University of California Lawrence Livermore National Laboratory

7000 East Ave.
P.O. Box 808
Livermore, CA 94550
Attn: R. Klein, MS-L023
Attn: J. Bolstad, MS-L023
Attn: C. Mailhiot, MS-L055
Attn: C. K. Nitta, MS-L096
Attn: T. F. Adams, MS-L098
Attn: R. McCallen, MS-L098

Attn: C. F. MacMillan, MS-L098
Attn: E. Dube, L-098
Attn: P. Raboin, MS-L125
Attn: R. W. Logan, MS-L125
Attn: D. Nikkel, MS-L342
Attn: S. Lee, MS-L560
Attn: S. F. Ashby, MS-L561
Attn: D. L. Brown, MS-L561
Attn: P. N. Brown, MS-L561
Attn: J. F. Mcenerney, MS-L591

Sandia Internal Distribution

MS 0429 J. S. Rottler, 02100
MS 0746 D. G. Robinson, 5411
MS 1201 J. M. McGlaun, 5903
MS 0449 G. M. Pollock, 6237
MS 1137 A. L. Hodges, 6534
MS 1137 G. K. Froehlich, 6535
MS1138 E. Shepherd, 6533
MS 0778 G. Barr, 6851
MS0735 T. Corbet, 6115
MS 9403 M. Baskes, 8712
MS 9405 R. Jones, 8742
MS 9405 P. Klein, 8742
MS 9405 R. A. Regueiro, 8743
MS 9011 J. C. Meza, 8950
MS 9011 P. T. Boggs, 8950
MS 9011 M. L. Koszykowski, 8950
MS 1110 L. J. Lehoucq, 8950
MS 0841 P. Hommert, 9100
MS 0828 T. C. Bickel, 9100
MS 0835 D. K. Gartling, 9100
MS 0836 M. R. Baer, 9100
MS 0836 C. W. Peterson, 9100
MS 0828 T. Y. Chu, 9100
MS 0841 J. A. Fernandez, 9102
MS 0828 R. K. Thomas, 9102
MS 0828 R. A. Garber, 9102
MS 0835 S. N. Kempka, 9111
MS 0835 S. W. Bova, 9111
MS 0835 S. Burns, 9111
MS 0835 A. A. Gossler, 9111
MS 0835 R. J. Cochran, 9111
MS 0835 B. Hassan, 9111 (5)
MS 0835 R. R. Lober, 9111
MS 0835 P. A. Sackinger, 9111
MS 0835 C. Roy, 9111
MS 0835 W. Wolfe, 9111
MS 0834 A. C. Ratzel, 9112
MS 0826 W. Hermina, 9113
MS 0826 T. J. Bartel, 9113
MS 0834 J. Johannes, 9114
MS 0834 K. S. Chen, 9114
MS 0834 L. A. Mondy, 9114
MS 0834 R. R. Rao, 9114
MS 0834 P. R. Schunk, 9114
MS 0825 W. H. Rutledge, 9115
MS 0825 F. G. Blottner, 9115
MS 0825 D. W. Kuntz, 9115
MS 0825 J. L. Payne, 9115
MS 0825 M. McWherter-Payne, 9115
MS 0825 D. L. Potter, 9115
MS 0825 K. Salari, 9115 (20)
MS 0836 E. Hertel, 9116
MS 0836 P. E. DesJardin, 9116
MS 0836 S. Tiezen, 9116
MS 0827 R. E. Hogan, 9117
MS 0827 R. O. Griffith, 9117
MS 0835 J. S. Peery, 9121
MS 0847 S. W. Attaway, 9121
MS 0847 M. L. Blanford, 9121
MS 0847 M. W. Heinstein, 9121
MS 0847 S. W. Key, 9121
MS 0847 G. M. Reese, 9121
MS 0555 M. W. Garrett, 9122
MS 0847 H. S. Morgan, 9123
MS 0847 J. B. Aidun, 9123
MS 0847 A. F. Fossum, 9123
MS 0847 D. R. Martinez, 9124
MS 0847 K. Alvin, 9124
MS 0847 J. Dohner, 9124
MS 0557 T. J. Baca, 9125
MS 0553 R. A. May, 9126
MS 0827 J. D. Zepper, 9131
MS 0827 G. Sjaardema, 9131
MS 0827 J. R. Stewart, 9131
MS 0827 H. C. Edwards, 9131
MS 0827 C. Forsythe, 9131
MS 0828 J. Moya, 9132
MS 0828 R. S. Baty, 9133

MS 0828 B. F. Blackwell, 9133
MS 0828 K. J. Dowding, 9133
MS 0828 A. R. Lopez, 9133
MS 0828 K. E. Metzinger, 9133
MS 0828 W. L. Oberkampf, 9133
MS 0828 T. Paez, 9133
MS 0828 V. J. Romero, 9133
MS 0828 C. Romero, 9133
MS 0828 A. Urbina, 9133
MS 0828 W. Withowski, 9133
MS 1135 D. B. Davis, 9134
MS 0321 W. J. Camp, 9200
MS 0318 G. S. Davidson, 9201
MS 0318 R. J. Pryor, 9201
MS 0316 J. E. Kelly, 9202
MS 0316 P. F. Chavez, 9204
MS 0819 T. G. Trucano, 9211
MS 1110 S. Chakerian, 9211
MS 0847 M. Eldred, 9211
MS 0847 A. A. Giunta, 9211
MS 1110 W. Hart, 9211
MS 1110 V. Leung, 9211
MS 1110 C. A. Phillips, 9211
MS 0847 J. Red-Horse, 9211
MS 0847 B. van Bloemen Waanders, 9211
MS 0318 P. Heermann, 9215
MS 1109 C. F. Diegert, 9215
MS 0316 S. S. Dosanjh, 9221
MS 1111 S. Plimpton, 9221
MS 1111 A. Salinger, 9221
MS 1111 J. N. Shadid, 9221
MS 1110 D. Womble, 9222
MS 1110 S. Istrail, 9222
MS 1110 R. Lehoucq, 9222
MS 0750 S. Minkoff, 9222
MS 1110 L. A. Romero, 9222
MS 1110 N. Pundit, 9223
MS 0321 A. L. Hale, 9224
MS 0321 J. Ang, 9224
MS 1109 R. Benner, 9224
MS 1109 J. L. Tompkins, 9224
MS 0316 G. Heffelfinger, 9225
MS 1111 H. Hjalmarson, 9225
MS 0847 R. Leland, 9226
MS 1111 B. Hendrickson, 9226
MS0847 P. Knupp, 9226 (10)
MS 0819 E. Boucheron, 9231
MS 0819 K. Brown, 9231
MS 0819 K. G. Budge, 9231
MS 0819 D. Carroll, 9231
MS 0819 R. Drake, 9231
MS 0819 A. C. Robinson, 9231
MS 0819 R. Summers, 9231
MS 0819 R. Weatherby, 9231
MS 0819 M. Wong, 9231
MS 0820 P. Yarrington, 9232
MS 0820 R. Brannon, 9232
MS 0820 D. Crawford, 9232
MS 0820 E. Fang, 9232
MS 0820 A. Farnsworth, 9232
MS 0820 M. E. Kipp, 9232
MS 0820 S. A. Silling, 9232
MS 0820 P. A. Taylor, 9232
MS 0419 R. G. Easterling, 9800
MS 0830 K. V. Diegert, 12323
MS 0829 M. Abate, 12323
MS 0829 B. M. Rutherford, 12323
MS 0638 M. A. Blackledge, 12326
MS 0638 D. E. Peercy, 12326
MS 0337 R. D. Skocypec, 15002
MS 1179 J. R. Lee, 15340
MS 0899 Technical Library,9616 (2)
MS 0612 Review and Approval Desk, 9612 (1) for DOE/OSTI
MS 9018 Central Technical Files,8940-2